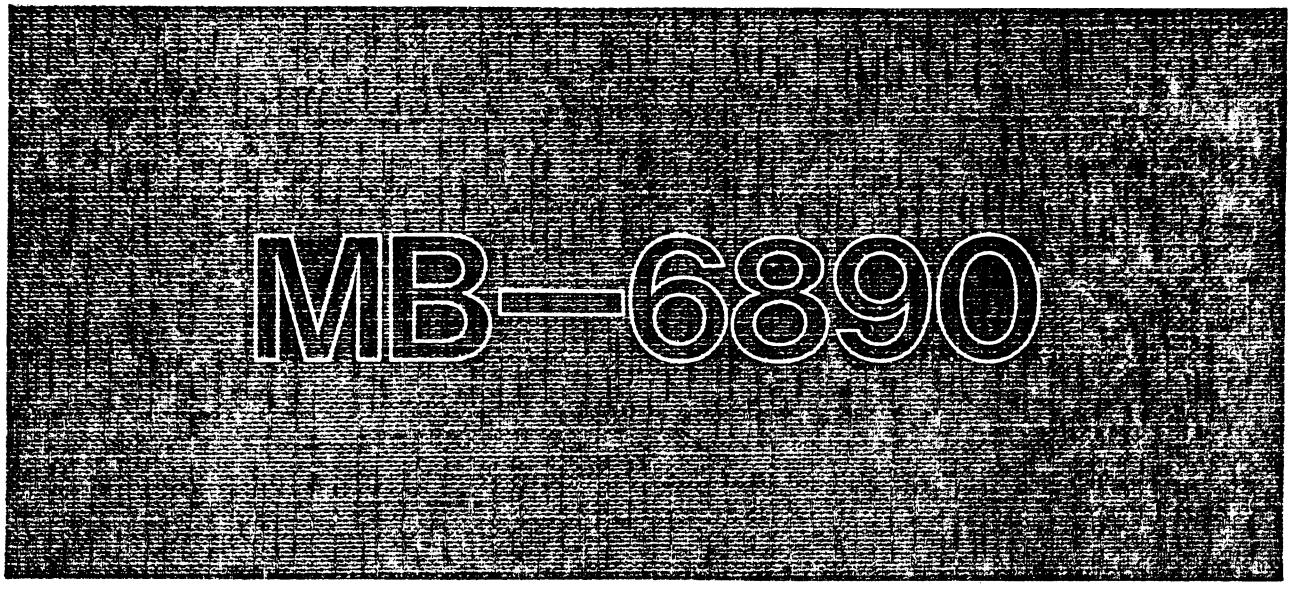
HITACHI

BASIC MASTER LEVEL-3



ASSEMBLER MANUAL

SALES DISTRIBUTOR:

1983

V.1.1

C O N T E N T S

CONTENTS (CONTINUED)

CONTENTS (CONTINUED)

1. CONCEPT OF ASSEMBLER AND DEBUGGER

This assembler is to convert your source program, your assembly language program fitted for HD-6809 of Basic Master Level-3 (MB-6890), into object program (machine codes) and to load the object program into the designated RAM memory location.

The assembly language is written with mnemonic codes which have 1 to 1 correspondence with the machine codes, and makes it easier to maintain your programs.

Together with the assembler, the debugger is also available so as to debug your object program. This debugger having editor commands to compile your source program makes it easier to develop your programs.

1. Necessary Hardware System Organisation

The assembler for Basic Master level-3 (MB-6890) is available either on cassette tape (MA-5100) or on diskette (MA-5200). MA-5100 works under the control of ROM BASIC through a cassette tape recorder as an I/O device and MA-5200 does under the control of DISK BASIC through a mini floppy disk unit as an I/O device.

In Table 1-1-1 there are shown the necessary hardware organisations both for MA-5100 and MA-5200.

COMPUTER AND DEVICE	MA-5100	MA-5200	EXPLANATION
1 Basic Master Level-3, MB-6890	0	0	N.B.1
2 Colour Display, C14-2170	0	0	C14-2170 or K12-2055P needed
3 Monochrome Display, K12-2055P			
4 Cassette Tape Recorder, TRQ-237	0		Usable for MA-5200
5 Mini Floppy Disk Drive, MP-3540	X	0	N.B.2
6 Dot-Matrix Impact Printer, MP-1040	0	0	Necessary to output assembled list

0 means: "Necessary".
means: "Possible to use".
X means: "Not needed".

Table 1-1-1 Necessary Hardware Organisation for Assembler

N.B.1 For Colour Monitor (C14-2170), the cable for the monitor (MP-9770) is needed.

N.B.2 For a mini floppy disk drive unit, I/F cards (MP-1800, and MP-1801) are needed. MA-5300 (for 32KB system) or MA-5301 (for 40KB system) is necessary as Basic Master Level-3 DISK BASIC.

N.B.3 The RAM expansion card (MP-9717) is necessary to use the MP-5301 Basic Master Level-3 DISK BASIC for the expansion of up to 40KB memory size.

N.B.4 The instructions of how to operate each I/O device are found in each manual.

2. Basic Master Level-3, and its Assembler and Debugger

Both on MA-5100 (cassette tape) and MA-5200 (diskette) are assigned Assembler and Debugger under the file name of ASM9, DEGUB5A, and DEBUD70, as shown in Table 1-2-1. DEBUG 5A and DEBUG 70 have exactly the same functions, but they are loaded at the different memory locations.

PROGRAM NAME	FILE NAME	MEMORY LOCATION
Assembler	ASM9	&H5000 &H6FFF
Debugger	DEBUG5A	&H5A00 &H69FF
	DEBUG70	&H7000 &H7FFF

N.B. The memory locations of ASM9 and DEBUG5A overlaps, so you cannot use both programs at the same time.

Table 1-2-1 Programs Written on MA-5100 and MA-5200

The memory size for Basic Master Level-3 is 32K bytes, as a standard orgainsation. If more memory size is needed, use the RAM expansion card which extends up to 40K bytes. If you expand the memory at the location of more than 9FFF in Hex, you cannot use the Assembler. Under the above mentioned conditions, a cassette tape recorder and a mini floppy disk unit are usable. MA-5100 is only used for a cassette tape device and MA-5200 used only for a mini floppy disk unit. The standard 32K bytes system organisation does not allow to use DEBUG70 program in case where a mini floppy disk drive is used as an I/O device, because Basic Master Level-3 DISK BASIC overlaps with the DEBUG70 memory area. Only in the above mentioned case, DEBUG5A has to be used instead of DEBUG70.

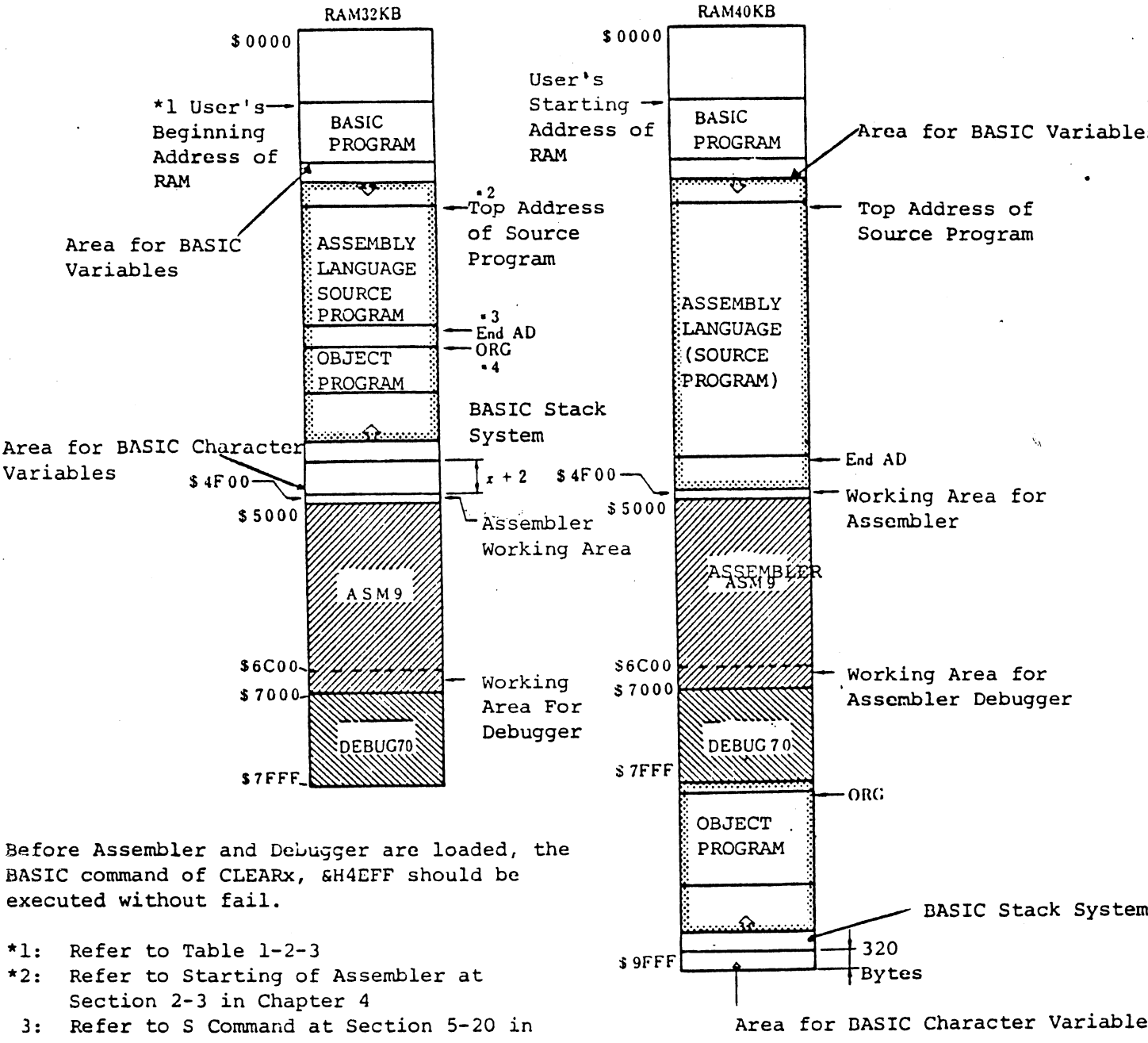
In Table 1-2-2 are shown the possible combinations of the available programs together with possible I/O devices.

RAM SIZE	FILE NAME	I/O DEVICES TO BE USED	
		CASSETTE TAPE RECORDER	MINI FLOPPY DISK
32KB	ASM9	◎	◎
	DEBUG5A	○	◎
	DEBUG70	◎	X
40KB	ASM9	◎	◎
	DEBUG5A	○	○
	DEBUG70	◎	◎

◎: Recommended combination
○: Possible combination
X: Impossible combination

Table 1-2-2 Possible Combination of Programs with I/O Devices

2-1 Memory Map

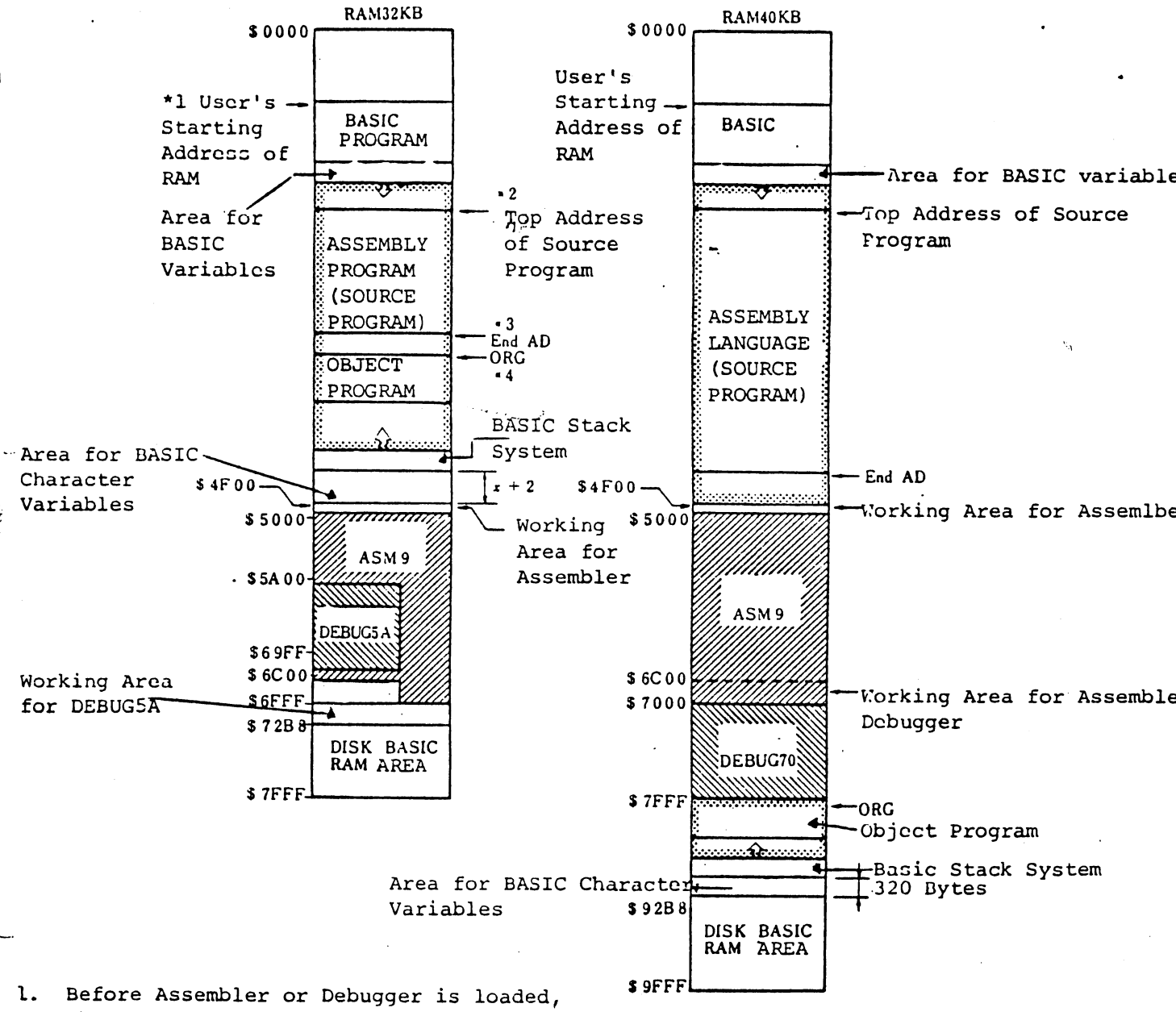


Before Assembler and Debugger are loaded, the BASIC command of CLEARx, &H4EFF should be executed without fail.

*1: Refer to Table 1-2-3
*2: Refer to Starting of Assembler at Section 2-3 in Chapter 4
3: Refer to S Command at Section 5-20 in Chapter 2
*4: Refer to Pseudo Commands of Assembler at Section 3 in Chapter 2

FIGURE 1-2-1 MEMORY MAP FOR MA-5100 (CASSETTE TAPE)

Figure 1-2-2 Memory Map for MA-5200 (Diskette)



1. Before Assembler or Debugger is loaded, the BASIC Command of CLEARx, &H4EFF should be executed without fail.
2. ASM9 and DEBUG5A have to be loaded alternately.
   The Above is valid for MA-5300 DISK BASIC

DISK BASIC MA-5300 can be used. In this case, DEBUG5A has to be used. The above is valid for MA-5301 DISK BASIC.

*1: Refer to Table 1-2-3
*2: Refer to Starting of Assembler at Section 3-1-3 in Chapter 4.
*3: Refer to S Command at Section 5-20 in Chapter 2.
*4: Refer to Pseudo Commands of Assembler at Section 3 in Chapter 2.

(1) The beginning address for users is different according to cassette based or disk based system as shown in Table 1-2-3.

Table 1-2-3 User's Beginning Address of RAM

SCREEN MODE	USER'S STARTING ADDRESS	
	MA-5100	MA-5200
40 Characters in normal mode	&HOB68	&HOF72
80 Characters in normal mode	&HOF68	&H1372
40 Characters in high-resolution mode	&H2768	&H2B72
80 Characters in high-resolution mode	&H4768	&H4B72

N.B.1 The above mentioned starting addresses are valid under the organisation. Different I/F cards such as printer or RS-232C cards will affect user's starting address. If you change FCB by using CONFIG program on DISK BASIC, the starting address will be greatly affected. For each case, refer to each detailed manual on how the starting address will be affected.

N.B.2 As for the setting of screen mode, refer to the manual on Basic Master Level-3 BASIC or DISK BASIC.

(2) The memory allocations are needed for Assembler and Debugger, as shown in Table 1-2-4.

Table 1-2-4 RAM size for Assembler and Debugger

PROGRAM NAME	FILE NAME	SIZE	RAM ADDRESS *1	WORKING AREA
Assembler	ASM9	8KB	&H5000-&H6FFF	&H4FOO-&H4FFF
Debugger	DEBUG5A	4KB	&H5AOO-&H69FF	&H6COO-&H6FFF
	DEBUG70	4KB	&H7000-&H7FFF	&H6COO-&H6FFF *2

*1: The memory allocations for ASM9 and DEBUG5A are overlapped. Therefore, you cannot use these at the same time.

*2: When you use both ASM9 and DEBUG70 at the same time, the working area for DEBUG70 overlaps with the memory allocation of ASM9, but the software design is made to share these memory areas by ASM9 and DEBUG70.

(3) Your BASIC program will be loaded, beginning at the memory location as shown in Table 1-2-3.

(4) The conversion area for BASIC characters is reserved automatically as shown in Talbe 1-2-5 when Basic Master Level-3 is switched on.

Table 1-2-5 BASIC Character Conversion Area after Power-on

	RAM SIZE	UPPER ADDRESS FOR BASIC PROGRAM	STARTING ADDRESS FOR BASIC CHARACTER CONVERSION AREA
ROM BASIC	32KB	&H7FFF	&H7ED1
	40KB	&H9FFF	&H9ED1
DISK BASIC	32KB	&H72B7	&H7189
(MA-5300 & MA-5301)	40KB	&H92B7	&H9189

After you have switched on Basic Master Level-3, you can set the starting address by CLEAR Command.

(5) The area for the BASIC variables and system stack will be assigned only by running your BASIC program. These are sizes are totally dependent upon your BASIC program.

(6) The BASIC FRE function value is given by the following formula.

FRE (0) = (Starting Address for BASIC Character Conversion)
- (Upper Address for BASIC Variable Area + 1)
- 14 *_____(1)

*: Number of bytes to be occupied in executing directly PRINT FRE (0)

When you extend your memory size up to 40KB and FRE function value exceeds 32767 (7FFF in HEX), the system will display the negative decimal value, "_N". In this case only the number of bytes will be given by equation (2).

FRE (0) = 65536 - N_____(2)

(7) The source program of the assembly language will be stored at any location designated by a user. But you can not designate this address which is lower than user's RAM starting address.

End AD (Last Address for Storage of Your Source Program + 1) can be known by executing Editor command "S".

(8) Object program (machine language) can be loaded sequentially at the RAM locations designated by ORG command.

(9) In case of DISK BASIC (MA-5300 and MA-5301), the memory area will be occupied as DISK BASIC RAM as shown in Fig. 1-2-2.

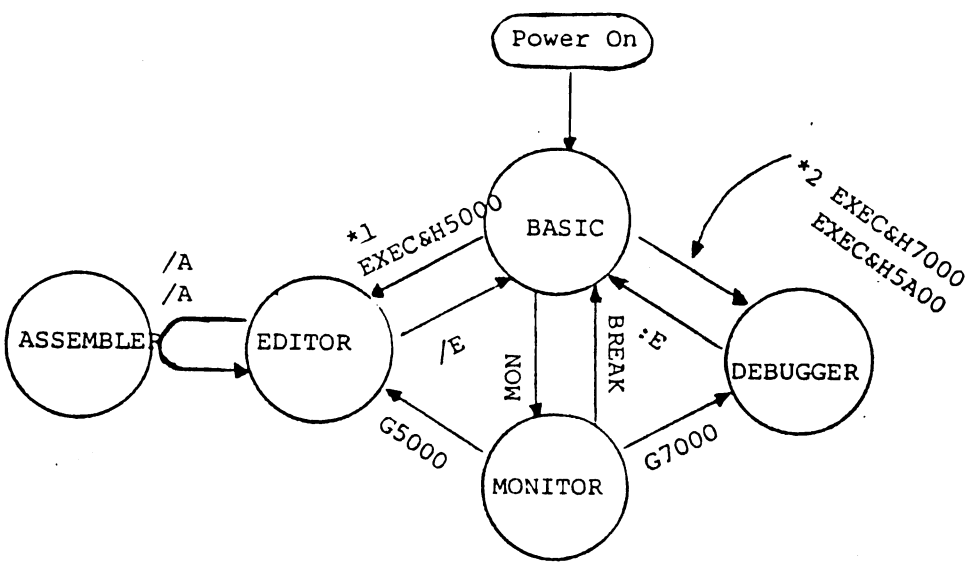
2-2 Transfer of Modes

When you load Assembler and Debugger in Basic Master Level-3, the following 5 different modes exist:

1. BASIC Mode
2. Monitor Mode
3. Editor Mode
4. Assembler Mode
5. Debugger Mode

Figure 1-2-3 shown how to transfer between 5 different modes.

Table 1-2-6 shows how to load Assembler and Debugger.



*1: LOADM "<Device Name> : ASM9",, R is also possible.

*2: LOADM "<Device Name> : DEBUG70",, R or LOADM" <Device Name> : DEBUG5A",, R is also possible.

Figure 1-2-3 Transfer between Different Modes

Table 1-2-6 Method of Loading Assembler and Debugger

ASSEMBLER/DEBUGGER	FILE NAME	OPERATION
Assembler	"ASM9"	LOADM"<Device Name> :ASM9" RETURN
		LOADM"<Device Name> :ASM9",,R RETURN
Debugger	"DEBUG5A"	LOADM"<Device Name> :DEBUG5A" RETURN
		LOADM"<Device Name> :DEBUG5A",,R RETURN
	"DEBUG70"	LOADM"<Device Name> :DEBUG70" RETURN
		LOADM"<Device Name> :DEBUG70",,R RETURN

N.B. As for device names, refer to the manuals on Basic Master Level-3 BASIC or DISK BASIC.

2. ASSEMBLER

1-1 Assembler Organisation

Assembler (File Name:ASM9), is designed based on the fundamental conecpts. The following are the basic features:

1) This assembler having the conversational editor makes it easy to make and compile source programs and to transfer the programs to I/O devices and retrieve from them. This has the function of printing Assembler listing on a printer.

2) The size of your source program which can be assembles at one time is 999 lines in maximum.

3) It is possible to develop your programs by using both Assembly language and BASIC language. In this case both languages can be stored in RAM without being over-lapped.

1-2 Storage area

1) the size of assembler is 8KB and it will be stored at the locations of &H5000 to &H6GGG.

2) The address area of &H4F00 to &H4FFF can be used as working area. Therefore, the memory area shown in figure 2-1-1 cannot be used for other purposes. The memory allocation with Debugger and BASIC should be referred in detail to Section 2-1 in Chapter 1 called Memory Map.

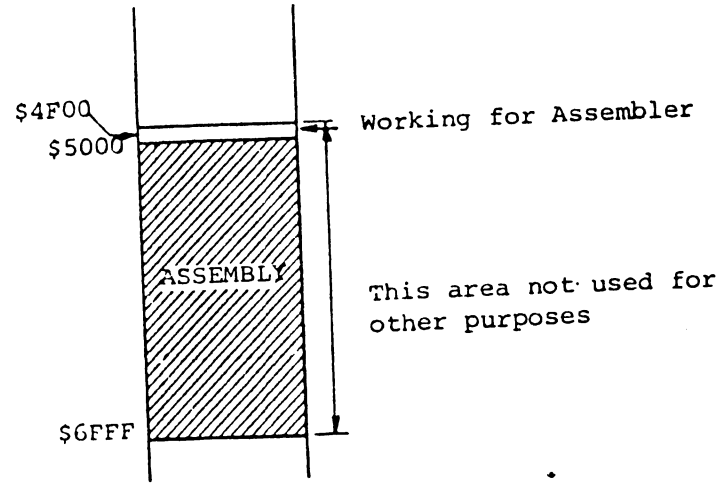


Figure 2-1-1 Storage Area for Assembler

2. Description of Assembly Language

2-1 Legal Letters and Symbold

The legal letters and symbols which can be used in your source programs are in the following:

1) Numerals

   0, 1, 2, ..., 9 (10 in all)

2) Letters

   A, B, C, ..., Z (26 in all)

3) Special Symbols are described below:

   `*` : When this is at the beginning of each line, the line becomes comments. When it is found in an operand, it shows relative address in reference to the line.

   `#` : This designates immediate address.

   `%` : Numerals following % are binary numbers.

   `&H,$`: Numerals and Letters (A to Z) following &H and $ stand for hexadecimal numbers.

   `<`: An operant following is of one byte.

   `>`: An operand following is of two bytes.

   `{}`: Two bytes bracketed {} are actual exectuable address.

5) Other miscellaneous

   When more than one space or TAB (CTRL + I) is inserted in the comments or operands, you can use numerals, Katakana, Hiragana, alphabet, special symbols and Kana symbols.

2-2 Source Program

An expression having meaning itself is called a statement. Writing a series of statements for your own purposes consists of your source program. Source program described in this manual has to be written according to the rules of Assembler language of HD6809.

2-2-1 Statements

Statements are divided into 3 as shown in figure 2-2-1.

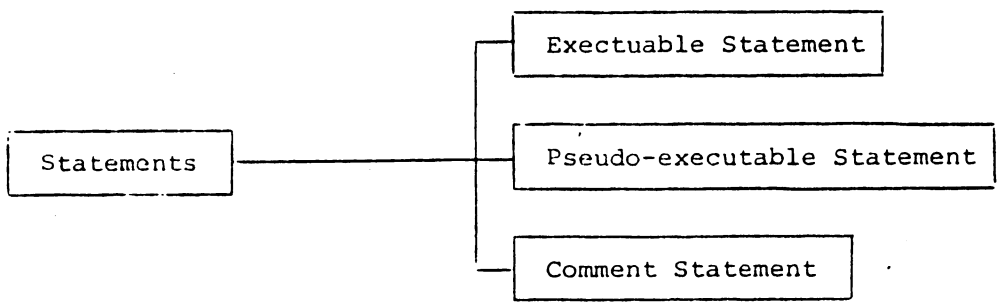


Figure 2-2-1 Classification of Statements

The functions of the 3 statements are describes in Table 2-2-1

STATEMENT	FUNCTION
Executable Statement	This is written by mnemonic codes having 1 to 1 correspondence with machine codes. 59 different basic statements are the most important executable statements.
Pseudo-Executable Statement	This is a statement to control assembly language conversion and is not converted to machine codes. There are 8 important pseudo-executable statements.
Comment Statement	To make it easy to follow up your program, comment statements are used. These are not converted into machine codes. These are printed on the assembly language list. By writing * at the beginning of a line, the following statements afterwards become comments.

Table 2-2-1 Classification of Statements

Among the above-mentioned statements, executable and pseudo-executable statements consist of 4 fields as shown in 2-2-2.

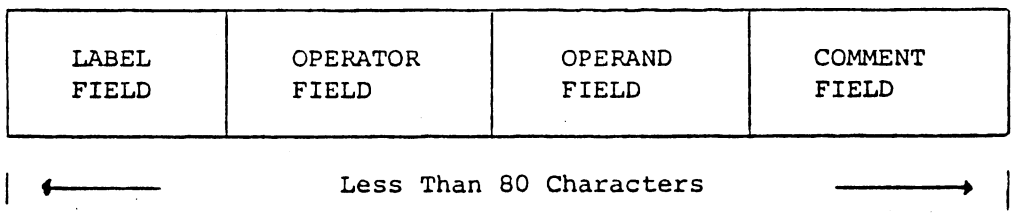


Figure 2-2-2 Organisation of Statement

A statement usually consists of 4 fields in a line, but only operator field or operator and operand fields can be found. More than one space or TAB symbol has to be inserted between each field, and at the end of each statement "RETURN" (&HOD) has to be inserted. Within each statement you can use 80 characters in maximum.

2-2-2 Label Field

In the label field, a label has to be used. A lable is used to refer to a statement by an instruction.

1) A label has to consist of equal to or less than 6 characters (including alphabet and numerals) beginning with the alphabet.

2) Register Names shown in Table 2-2-2 or "PCR" cannot be used as label names. EOF or label names having EOF cannot be used either.

3) In the same program, lable names have to be defined consistently.

4) The maximum number of labels to be used is 128. When your program exceeds 127, an error will take place.

5) A label has to be started at the beginning of each line. If you insert a space at the beginning, the system understands that this is an operator.

6) You can assign a label to an executable statement, but you cannot do a label to a pseudo-executable statement except for a few. Refer to pseudo-executable statements in Chapter 3 for details.

REGISTER NAMES	SYMBOL	REGISTER NAMES	SYMBOL
Accumulator A	A	X-Index Register	X
Accumulator B	B	Y-Index Register	Y
Double Accumulator	D	Hardware Stack Pointer	S
Condition Code Register	CC	User Stack Pointer	U
Direct Page Register	DP	Program Counter	PC

Table 2-2-2 Table of Registers

2-2-3 Operator Field

Commands legal to operator fields are of 2 kinds as follows:

1) Mnemonic codes of HD6809. Refer to Table 2-2-3.

2) Pseudo-executable commands. Refer to assembler pseudo-executable commands in Chapter 3.

TABLE 2-2-3 CLASSIFICATION OF HD680 9 MNEMONIC CODES

	FUNCTION	COMMANDS
1.	Addition and Subtraction concerning 8-bit Registers	ADC, ADD, SBC, SUB
2.	Addition and Subtraction concerning 16-bit Registers	ADDD, SUBD, ABX
3.	Multiplications (between 8-bit Registers)	MUL
4.	Comparison and Test	CMP TST
5.	Shift Rotate	ASL, ASR, LSL, LSR, ROL, ROR
6.	Logical Arithmetic	AND, BIT, COM, NEG, EOR, OR
7.	Load and Store	LD, ST, PSH, PUL
8.	Branch over Short Range	BCC, BCS, BEQ, BGE, BGT, BHI, BHS, BLE, BLO, BLS, BLT, BMI, BNE, BPL, BVC, BVS
	Branch over Long Range	LBCC, LBCS, LBEQ, LBGE, LBGT, LBHI, LBHS, LBLE, LBLO, LBLS, LBLT, LBMI, LBNE, LBPL, LBVC, LBVS
9.	Unconditional Branch	BRA, LBRA, BRN, LBRN, JMP, NOP
10.	Subroutine Control	BSR, LBSR, JSR, RTS
11.	Interrupt Commands	SWI, SW12, SW13, RTI, SYNC, CWAI
12.	Transfer between Registers	EXG, TFR
13.	Load Command of Actual Address	LEA

	FUNCTION	COMMANDS
14.	Sign Transaction Command between 8 and 16 bits Conversion	SEX

N.B. As for HD6809 Mnemonic codes, refer to Chapter 5 for details.

2-2-4 Operand Field

In an operand field, coded information to be executed is written. Depending on instructions, operand fields are not always necessary. Depending on instructions, multiple information is needed. In this case a comma separates two different kinds of information.

In operand fields, legal formats are allowed as shown in Table 2-2-4.

Table 2-2-4 Formats of Operand Fields

FORMAT	CLASSIFICATION	EXPLANATION
DATA FORMAT	Numeral Constants	Decimal Number, Hexadecimal Numbers (Followed by $ or &H), Binary Numbers (Followed by %)
	Character Constants	/Character strings/
	Symbols	Symbols (Label Names) or *(Relative Address)
	Equation	Numeral constants and symbols are allowed as terms of an equation. Arithmetic operators (+ and -) are allowed as operators of an expressions.
ADDRESS FORMAT	There are 8 different address formats of instructions. Refer to Address Format at Section 2 in Chapter 5 for details.	

An example is shown below

Memory Address	Data	Label	Operator	Operand
	000A	ABC	EQU	10
	0002	BB	EQU	2
			ORG	&H3000
3000	960C		LDA	ABC+BB
3002	D608		LDB	ABC-BB
3004	960C		LDA	ABC+2
3006	D608		LDB	ABC-2
3008	B63000A		LDA	*+BB
300B	F630001		LDB	*-10
300E	B63000E		LDA	10+*-ABC
3011	EF		FCB	&HEF, ABC
3012	0A			
3013	A60A		LDA	ABC, X
3015	54		FCC	/TEXT/
3016	45			
3017	58			
3018	54		END	

2-2-5 Comment Field

A comment field starts in an operand field by inserting more than one space or TAB symbol at the beginning. In comment fields, you can use numerals, Katakana, Hiragana, Alphabet, symbols, and Kana symbols that can be found on the keyboard of Basic Master Level-3. By using these you can state any comments as you like. Comments are not converted into machine codes in assembling your source program. They are output on your assemble list. Therefore, these make it easy for you to follow up your program.

3. ASSEMBLER PSEUDO-EXECUTABLE COMMANDS

Assembler psuedo-executable commands are the ones to control Assembler, and they are not converted into machine codes. Pseudo-executable commands are described in operator fields. A few such commands needs labels and operands, and some do not need them.

1) END (END of Program)

(1) This is an pseudo-executable command to show the end of your program (Statements after "END" will never be assembled at all).

(2) You do not allow to use labels or operands.

2) EQU (EQUate Symbol)

(1) This is to assign numbers to labels

(2) You can not omit labels nor operands.

Example:

LABEL	OPERATOR	OPERAND
START	EQU	&HAOO
AB	EQU	10
AC	EQU	AB
XY	EQU	YZ + AB
YZ	EQU	&HFOO

3) RMB (Reserve Memory Byte)

(1) Memory bytes are reserved by the value of an operand.

(2) You can omit labels. But operands will never be omitted.

Example:

Memory Address	Label	Operator	Operand
3000		RMB	2
3002	DATA1	RMB	10
300C	DATAX	RMB	1

4) FCB (Form Constant Byte)

(1) 8-bit data are sotred in memory area in correspondence to operands. Multiple operands are separated by commas.

(2) When illegal characters found, the system stores zeros automatically.

(3) You can omit labels. But operands will never be omitted.

4) FCB (Form Constant Byte) (Continued)

Example:

Memory Address	Data	Label	Operator	Operand
3000	FF	XYZ	FCB	&HFF
3001	AA	ZXY	FCB	&HAA, 12,
3002	OC			
3003	OO			

5) FDB (Form Double Fyte)

(1) 16-bit data are loaded into memory area according to operands. Multiple operands are separated by commas.

(2) When illegal characters found, the system loads 0 there.

(3) Lables can be omitted, but operands cannot.

Example:

Memory Address	Data	Label	Operator	Operand
3010	0001	ONE	FDB	1
3012	0000	DATA	FDB	,&HABC,&HF
3014	OABC			
3016	OOOF			

6) FCC (Form Constant Characters)

(1) This is to convert into JIS codes and to store in the memory location.

(2) Legal characters are from $20 to $FE in JIS codes.

(3) The system understands bracketed characters by 2/'s as legal ones.

(4) Labels can be omitted, but operands cannot.

Example:

Memory Address	Data	Label	Operator	Operand
3000	42	MESSI	FCC	/BASIC/
3001	41			
3002	53			
3003	49			
3004	43			

7) ORG (ORGin)

(1) This pseudo-executable commands designates the starting address of the object program when your source program is converted into machine codes.

7) ORG (ORGin) (Continued)

(2) ORG command can be used several times in the same program.

(3) Labels cannot be used. Operands cannot be omitted.

8) SETDP (SETDirect Page Pseudo Register)

(1) This command assign a value to the Direct Page Pseudo Register, and tells the Assembler on the range (page) of the direct address format.

(2) When you do not designate the page address*, the Assembler will set the value of Direct Page Pseudo Register to be zero by default.

(3) The content of the Direct Page Pseudo Register set by SETDP command is valid until the content of the register is re-assigned by the new SETDP command.

(4) The Direct Page Register (DPR) of HD6809 Microprocessor is not changed by SETDP command. Therefore, the content of DPR has to be set as the same value as that of Direct Page Pseudo Register by TFR or EXG command.

(5) Labels cannot be used. Operands cannot be omitted.

Example:

Memory Address	Data	Label	Operator	Operand
4001	0064	BUF	RMB	100
	0040		SETDP	&H40
4065	8640		LDA	#H40
4067	1F8B		TFR	A, DP
4069	8610		LDA	#&H10
406B	9701		STA	BUF

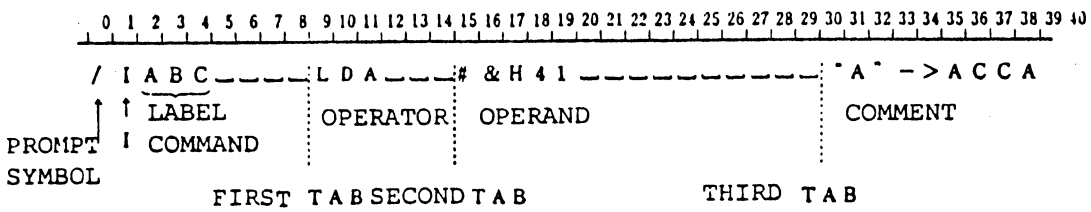
* As for the page address, refer to the direct address format at Section 2-4 in Chapter 5.

4. MISCELLANEOUS

The rules to describe the source program of the assembly language was described at Section 2 of "Description of Assembly Language" and Section 3 of "Assembler Pseudo-Executable Commands" in this Chapter. Additional rules are to be described further at this section.

4-1 TAB Symbol (CTRL + I , $09)

This TAB symbol is inserted between a label and an operator, between an operator and operand, and between an operand and a comment. Therefore, each field is adjusted properly. When your source program is assembled, this TAB symbol is neglected. When the system loads your source program, this occupies one byte for the storage. The TAB position is fixed as shown in Figure 2-4-1. In Figure 2-4-1 is shown the screen format of your source program on the CRT display.



N.B. One space is automatically inserted when the TAB Symbol is used after the 30th column.

Figure 2-4-1 Position of TAB

4-2 Description of Operator

No space is needed between the mnemonic symbol and register name. If a space is placed, an error takes place when assembled.

Example:

WRONG DESCRIPTION	RIGHT DESCRIPTION
LD␣A	LDA
ST␣U	STU
CMP␣D	CMPD
PSH␣U	PSHU

4-3 Description of Index Address Format

An operand of index address format is described as follows:

Expression , R R:Pointer Register

When the expression takes the value of 0, you can omit "0" or ",". You can just write the register name only.

Example:

LDA X
LDA ,X } In each case they have the same meaning of LDA o,X.

The same rule will be applied to automatic increment and automatic decrement.

4-4 Reference Definition of Label Name

It is possible to make a reference to and define a previous statement. In case of the reference to more than 2 statements, the date becomes zero.

Example:

Memory Address	Data	Label	Operator	Operand
	0041	L1	EQU	L2
	0041	L2	EQU	&H41 ← Refer to one previous statemen
	0044	L3	EQU	L2+3

If before a statement defining a label there is a statement referring to this label, the particular statement referring to the label is treated as extended address format or as 16-bit offset.

Example:

Memory Address	Data	Label	Operator	Operand
	0041	L1	EQU	L2
	0041	L3	EQU	&H41
			ORG	&H3000
3000	B60041		LDA	L1
3003	9641		LDA	L2
3005	A6890002		LDA	L3, X
	0002	L3	EQU	2

4-5 Forced Direct and Forced Extended Address Format

By putting ""or"" at the beginning of an operand, the operand value becomes one byte or two bytes. The address formats applying forced direct or forced extended address formats are in the following:

(1) Direct Address Format

(2) Extended Address Format

(3) Extended Indirect Address Format

(4) Index Address Format (but no applicable to Accumulator Offser, Auto-increment, or Auto-decrement)

1) When Forced Extended Address Format ( ) is applied to Direct Address Format, the system assembles as Extended Address Format.

2) When Forced Direct Address Format ( ) is applied to Extended Address Format, the lower one byte of an operand becomes valid and the system assembles as Direct Address Format. When in this case the higher one byte of an operand is different from the value of the Direct Page Pseudo Register, the effective address is different by using Forced Direct Address Format. Therefore, an error takes place when assembled.

3) When Forced Direct Address Format ( ) is applied to Extended Indirect Address Format, the higher one byte is assembled as "$00".

4) When Forced Extended Address Format ( ) is applied to Index Address Format, the system assembles as 16-bit offset. When Forced Direct Address Format ( ) is applied, the lower one byte of an operand becomes valid and the system assembles as 8-bit offset. Except that the higher one byte of an operand is "$00", the system will issue an error message. In case of 0-bit offset and 5-bit offset, Forced Direct Address Format ( ) is neglected.

Example:

Memory Address	Data	Label	Operator	Operand
	0040		SETDP	&H40
4000	B64001		LDA	>&H4001
4003	9612		LDA	<LABEL1
4005	A69F0056		LDA	>[LABEL2]
4009	A6890001		LDA	>1, X
400D	A68856		LDA	<LABEL2, X
	4012	LABEL1	EQU	&H4012
	0056	LABEL2	EQU	&H56

N.B. When Forced Direct Address Format (<) or Forced Extended Address Format (>) is bracketted by [] for Extended Indirect Address Format, it works in the same way as in the above example.

4-5 Forced Direct and Forced Extended Address Format

4) (Continued)

Example:

```
LDA     [<LABEL]
STA     <[LABEL]
```

5. EDITOR COMMANDS

When the system works under the Assembler mode, it waits for editor commands. Inputting editor commands makes the system edit your files and assemble your source program.

This Assembler understands that your source program consists of each statement on each line, and editing working is executed by designating the particular line to be modified.

There are 21 editor commands in all. They are functionally divided in the following:

(1) Moving the particular line to be modified

(2) Insertion and deletion

(3) Listing, searching and exchanging

(4) Filing, Verifying, and merging

(5) Assembling

(6) Others

The assembling command is treated as one of editor commands, and the system waits for the next editor command as soon as the assembling work is finished. Therefore, the modification can be easily made. This also makes it easy to work your program.

In Table 2-5-1 is shown the table of editor commands. In Section 5-1 and thereafter each command is described in detail.

Table 2-5-1 Table of Editor Commands

CLASSIFICATION	NO	COMMAND	FORMULA	FUNCTION
Moving the line to be modified	1	n command	n [RETURN]	The system moves nth line to be modified, and displays the content of the line on the screen.
	2	/ command	/n [RETURN]	The system moves the nth line in advance, and displays the content of the line on the screen.
	3	∧ command	n [RETURN]	The system moves the nth line beforehand, and displays the content of the line on the screen.
Insertion and Deletion	4	"Space" Command	"Space" Characters [RETURN]	The system adds the input characters after the end of the line to be modified and this becomes the new line.
	5	I command	I Characters [RETURN]	The system adds the input characters at the beginning of the previous line to be modified, and the previous line becomes the one to be modified.
	6	K command	Kn [RETURN]	The system kills the n lines including the line to be modified. The next line becomes the one to be modified. The content is displayed on the screen.
	7	L command	Ln [RETURN]	The system displays the contents of the n lines including the present line to be modified. Afterwards, the next line becomes the one to be modified.

CLASSIFICATION	NO	COMMAND	FORMULA	FUNCTION
Listing, Searching, and Modifying	8	F command	F characters RETURN	The system searches the same character pattern from the line to the last line. The first line where the same input character pattern is found becomes the present line to be modified. The content of the line is displayed on the screen.
	9	P command	P RETURN	The content of the present line to be modified is displayed on the screen.
	10	P command	P Characters RETURN	Some part of the line to be modified is replaced with the input characters (line editing).
	11	X command	X Previous Characters/New Characters RETURN	Within the line to be modified, the system searches the previous characters designated and the previous ones are replaced with the new ones if found. The new line is displayed on the screen. If the system cannot find the match, nothing is displayed on the screen.
	12	Y command	Y Previous Characters/New Characters RETURN	X command is executed from the line to be modified to the last line. Only replaced lines are shown on the screen.
Saving, Verifying, and Merging	13	S command	S "Program Name" RETURN	The system saves your program from the first line to the line together with the program name either on cassette tape or on diskette.
	14	V command	V "Program Name" RETURN	The system verifies that the check sum of your source program stored on the RAM is the same as that on cassette tape.

CLASSIFICATION	NO	COMMAND	FORMULA	FUNCTION
	15	M command	M "Program Name" RETURN	The program on cassette tape or on diskette is merged after the last line to be modified
Assembling	16	A command	A RETURN OR A, RETURN	Assembling of your program is executed and the assembled list is displayed on the screen. In case of A, RETURN, list is displayed one screen by one.
	17	AL command	AL RETURN	Assembled list is output on a printer.
Others	18	N command	N RETURN	The system displays the number of the present line to be modified.
	19	C command	C "New Prompt Character" RETURN	The previous prompt mark is replaced with the new one designated in the command. This prompt mark is used as the demarcation code for P, X, and Y, commands.
	20	S command	S RETURN	The system displays the last address of your source program plus one. The system displays as follows: End AD = $XXXX
	21	E command	E	The system returns to the BASIC mode.

5-1 n Command

1. Formula n RETURN (n: Positive integer)

2. Function

The system moves the line to be modified, and the content of the line is displayed on the screen.

3. Operation

(1) While the system is waiting for an editor command, you can key in the desired number. If you make a mistake in keying in, you can delete the numbers by using [DEL] key. You can key in the new numbers.

(2) When you hit the [RETURN] key, the system moves to the designated nth line to be modified, and displays the content of the line. The system waits for the next editor command.

(3) The number of a line has to be designated within 3 decimal digits (1 to 999).

(4) Others

If you input the line number which exceeds the last line of your source program, the EOF line becomes the one to be modified. The system displays EOF.

N.B.1 The assembler can handle the 999 lines of your source program in maximum.

N.B.1 The system automatically inserts the EOF line (End of File) in the next line after the last line of your source program.

Example:

The Third line is the one to be modified.
And the content is displayed on the screen.

```
/L 4
  AA     LDX   WW
         DECB
  ABC    BNE   XY
         LDA   TE
  XY     JMP   ZZ
/3
  ABC    BNE   XY
/_
  ↑
Blinking Cursor
```

Prompt Mark →
Line to be Modified →

5-2 /Command

1. Formula /n [RETURN] (n: Positive integer)

2. Function
The system moves the line to be modified to the nth line in advance, and displays the content.

3. Operation

(1) While the system is waiting for an editor command, jar key in the number of the lines followed by "/" key which you want to move. If you make a mistake in keying in, you can delete by using the [DEL] key. You can key in the new numbers.

(2) When the [RETURN] key is pushed, the system moves to the nth line in advance to be modified. The content is shown on the screen. The system is waiting for the next editor command to come.

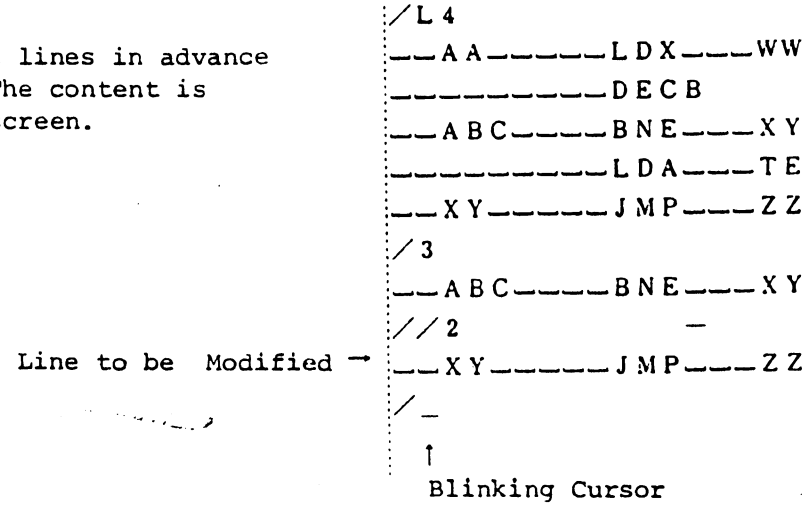
(3) The number has to be within 3 decimal digits (1 to 999).

(4) If the input number is 1, you can omit it.

4. When you want to move beyond the last line of your source program, EOF becomes the line to be modified.

Example:

The system moves 2 lines in advance to be modified. The content is displayed on the screen.



5-3 ∧ Command

1. Formula ∧ n [RETURN] (n: Positive integer)

2. Function

   The system moves n lines backwards to be modified and displays the content of the line.

3. Operation

   (1) When the system is waiting for an editor command, just key in the number of the lines followed by "∧" which you want to move. If you make a mistake in keying in, delete one character by one by using DEL key. you can key in the new numbers again.

   (2) When you [RETURN] key, the system moves n lines backwards to be modified. The content of the line is displayed on the screen. The system is waiting for another command.

   (3) The number has to be within 3 decimal digits (1 to 999).

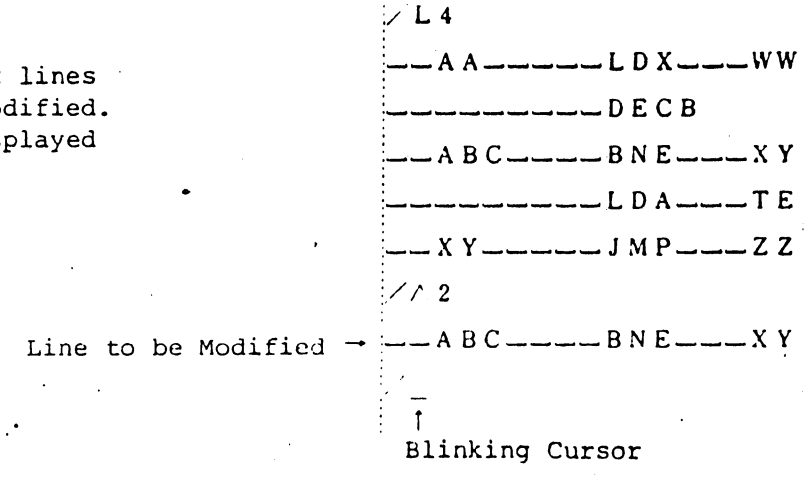
   (4) If the input number is 1, you can omit it.

4. Others

   If you want to move backwards beyond the beginning of your source program, the first line of your program becomes the one to be modified.

Example:

The system moves 2 lines backwards to be modified. The content is displayed on the screen.



5-4 "Space" Command

1. Formula (Space) Characters [RETURN] .

2. Function

   You can add the characters after the line to be modified. The modified line becomes the new one to be modified.

3. Operation

   (1) When the system is waiting for an editor command, you can key in the characters followed by the SPACE key which you want to add. If you make a mistake in keying in, you can delete one character by one by using [DEL] key. You can key in new characters.

   (2) When you key in [RETURN] key, the system adds the characters after the previous line to be modified. The modified line becomes the new one to be modified. The system does not display the modified one. It is waiting for another command.

   (3) When you hit [RETURN] key after [SPACE] key, the system fills spaces.

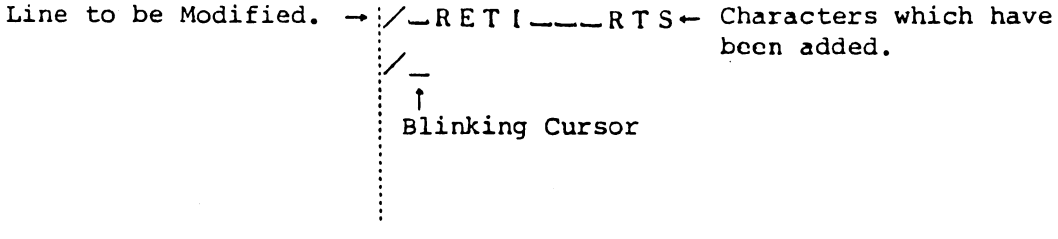
4. Others

   (1) When you apply this command after EOF, the characters are filled before EOF.

Example:

1. The characters that you want to add are keyed in, and this command has been executed.

2. To make sure that this command has been executed, go backwards by ^ command, and list the content of the line by L command.



5-5 I Command

1. Formula I Characters [RETURN] .

2. Function

   This command inserts characters before the line to be modified. The previous line is still the one to be modified.

3. Operation

   (1) When the system is waiting for an editor command, you can key in characters followed by [I] key which you want to insert. If you make a mistake in keying in, you can delete one character by one by using [DEL] key. You can key in new characters again.

   (2) When you hit the [RETURN] key, the system inserts the characters before the line to be modified. The previous line is still the one to be modified. The system does not display the modified line.

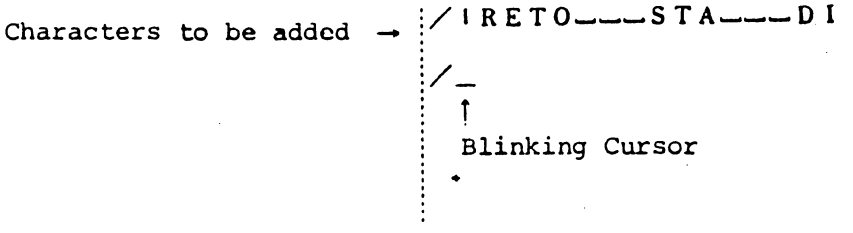
   (4) When you hit [RETURN] key right after [I] , the system fills spaces.

   N.B. The system only allows 80 characters in one line in maximum. Therefore, the system only accepts 80 characters for one line.

   Example:

   1. The characters that you want to insert have been keyed in, and I command has been executed.

   2. To make sure that the command has been actually executed, go backwards by command and list the content of the line by L command.



5-6 K Command

1. Formula Kn [RETURN] (n: Positive integer)

2. Function

This command deletes n lines including the line to be modified. The next line becomes the new one to be modified, and the system will display the content of the present line to be modified.

3. Operation

(1) When the system is waiting for an editor command, you can key in the number of lined followed by [K] key which you want to kill. If you want to make a mistake in keying in, you can delete one character by one by using [DEL] key. You can key in new number.

(2) When you hit the [RETURN] key, the system will kill n lines including the line to be modified. The next line becomes the new one to be modified. The content of the new line to be modified is displayed on the screen.

(3) The number has to be within 3 decimal digits (1 to 999).

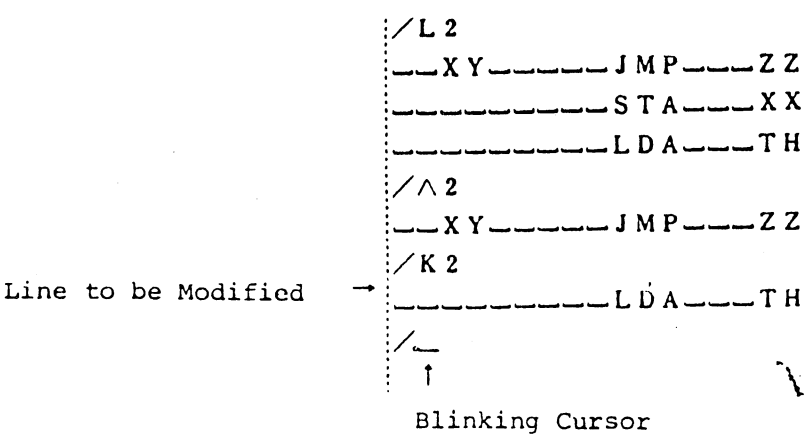
(4) If the number of lines is 1, you can omit it.

4. Others

When the number of lines to be killed exceeds the last line, the system will erase up to the last line. And EOF becomes the new line to be modified.

Example:

The 2 lines have been killed including the line to be modified. The next line will be the one to be modified, and the content of the new line to be modified is shown on the screen.



5-7 L Command

1. Formula Ln [RETURN] (n: Positive integer)

2. Function

This command displays the contents of the n lines including the line to be modified. The new line to be modified becomes the n + 1st line. Therefore, the new line to be modified is displayed on the screen.

3. Operation

(1) When the system is waiting for an editor command, you can key in the number of the lines followed by [L] key which you want to show on the screen. If you make a mistake in keying in, you can delete one character by one by using [DEL] key. You can key in the new value.

(2) When you key in [RETURN] key, the system displays the n lines on the screen including the line to be modified. The new line to be moved becomes the n + 1st line, and its content of the line is shown on the screen. The system is waiting for another command.

(3) The number has to be within 3 decimal digits (1 to 999).

(4) If the number of the line is 1, you can omit.

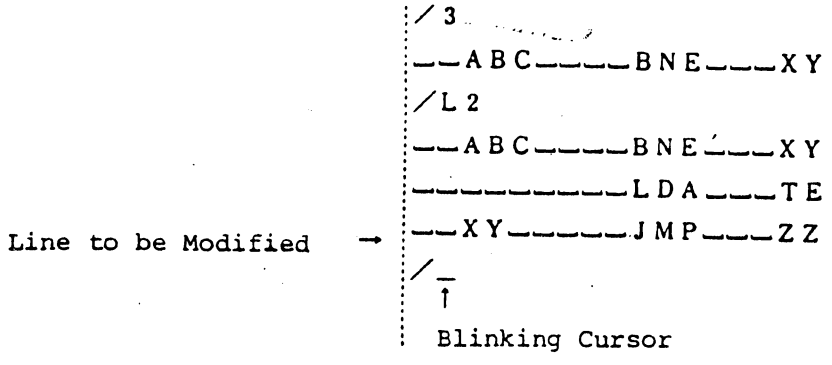
(5) If the number of the lines exceeds the whole screen, the system cannot display at one time. If in this case the system displays the first 21 lines, the display will be stopped. When you hit the [SPACE] key, the next lines will be shown on the screen. When the prompt mark appears on the screen, this means that the file reached the end.

(6) If the number of the lines that you want to show exceeds the last line, the system output up to EOF and the EOF Line becomes the line to be modified.

N.B. If you want to halt the display halfway, push the [RESET] switch. If you hit the [RESET] switch, the system returns to the BASIC mode. If you want to go back to the Assembler mode, type in EXEC&H5006. When you key in EXEC&H5006, the system is waiting for an editor command.

Example:

The 2 lines from the line to be modified are shown on the screen and the next new line is the present line to be modified. Therefore, the new line to be modified is displayed on the screen.



5-8 F Command

1. Formula F Characters [RETURN] .

2. Function

This command searches the same character pattern among the line to be modified up to the last line. If the system finds the same pattern in a line first time, the system understands the line as the new line to be modified and displays the line on the screen.

3. Operation

(1) When the system is waiting for an editor command, type in the character pattern followed by [F] key. If you make a mistake in keying in, you can delete one character by one by using [DEL] key. You can key in again.

(2) When you hit [RETURN] key, the system executes the F command and finds the matched line. It displays the line which becomes the new line to be modified. And the system is waiting for another editor command.

(3) If the system cannot find the matched line, the EOF line becomes the new line to be modified. The system shows the EOF line and is waiting for another command.

4. Others

When you want to search the match from the beginning of your source program to the end, put i in n command (This means that the line to be modified becomes the first line of your source program). And then use the F command.

N.B. The character pattern cannot occupy the different fields.

Example:

The character pattern of "RTS" to be searched has been keyed in, and the F command has been executed.

```
                    /FRTS
                    __RETO___RTS
Line to be modified →
                    /_
                     ↑
                    Blinking Cursor
```

5-9 P Command (1) (Print Command)

1. Formula P [RETURN]

2. Function

   This command shows the line to be modified. The line to be modified is not moved.

3. Operation

   (1) When the system is waiting for an editor command, key in P key. The system displays the line to be modified. The cursor is blinking at the beginning of the line to be modified.

   (2) When you hit the [RETURN] key, the system is waiting for another command.

   Example:

   The P command has been executed.

```
/P
--RET1---RTS
  ↑
  Blinking Cursor
  (When the [RETURN] key is hit, the system
  is waiting for another command)
```

5-10 P Command (PUT Command)

1. Formula P Characters [RETURN] .

2. Function

   The content of the line to be modified is modified.
   The line to be modified is not moved.

3. Operation

   (1) When you type in [P] key during the waiting for an editor command, the content of the line to be modified is displayed on the screen, and the cursor is blinking at the beginning of the line.

   The modification can be made effectively by using the following functions of the keys.

   1) key moves the cursor right.

   2) Key moves the cursor left.

   3) When the prompt mark or [INS] key is hit, the character pattern after and at the cursor position is shifted right in parallel and one space is inserted at the cursor position.

   4) [DEL] key deletes one character before the cursor position and the character pattern after the cursor position is shifted left in parallel.

   5) When the other keys than those mentioned in (1) to (4), the character at the cursor position is replaced with the input character.

   (2) The above mentioned operation can be repeated many times until you hit the [RETURN] key.

   (3) When you hit the [RETURN] key, instead of the previous line to be modified, the line after the modification is stored in the storage area of your source program.

   (4) The line to be modified is not moved.

   (5) When the number of characters in one line exceeds 80 by the above mentioned operations, the first 80 characters from the beginning are stored in the memory area of your source program.

4. N.B.

   (1) If you want to use the prompt mark ("/") as the input character, replace the prompt mark with the other symbol by the C command. Then, input the prompt mark by the P command.

(2) When you return the cursor to the beginning and hit the [RETURN] key, the P command is neglected. If the operation of the P command is messed up, start from the beginning by using this operation.

(3) The TAB symbol ([CTRL] + [I]) cannot be used under the P command.

5-11 X Command

1. Formula X Previous Characters [PROMPT MARK] New Characters [RETURN]
   The prompt character is "/" under the initial condition.

2. Function

   (1) Within the line to be modified, this command searches the previous characters and replaces with the new characters if found. The line to be modified is displayed on the screen after the replacement.

   (2) If the match is not found, the replacement is not executed. The line to be modified is not displayed on the screen.

   (3) The line to be modified is not moved.

3. Operation

   (1) When the system is waiting for an editor command, you should type in the character pattern to be replaced and then the new character pattern after the prompt mark ("/").

   (2) When the [RETURN] key is hit, the command is executed.

      (1) If the system has found the character pattern to be to be replaced, the new character pattern is replaced with. And the system displays the content of the line after the modification.

      (2) If the system cannot find the match, the replacement is not executed. The content of the line is not displayed, either.

      (3) The replacement can be executed without depending on the length of the character pattern to be replaced nor the new character pattern. But one line has to consist of within 80 characters.

      (4) If the system finds several character patterns in the same line to be modified, all of them are replaced with the new character pattern.

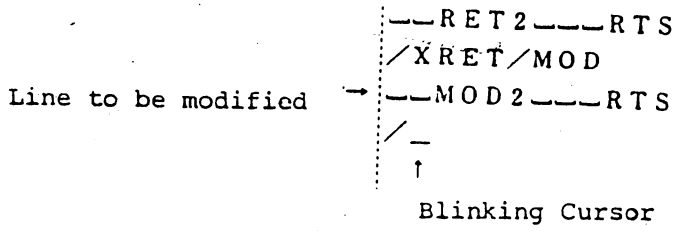
   (4) Others

      If you set one character as the character pattern to be modified, there is possibility that the system would find the character pattern at some other location where you do not want to replace. Therefore, it is desirable to use several characters as the pattern.

      N.B. The character pattern must not occupy the different fields.

Example:

The character pattern to be replaced has been keyed in, and the X command has been executed.



5-12 Y Command

1. Formula Y Character pattern to be replaced [PROMPT MARK] New Character pattern RETURN . The prompt mark is "/" under the initial condition.

2. Function

   This command is executed from the line to be modified to the last line. Only the lines where the replacements have taken place are displayed on the screen. On the last line EOF is displayed. This EOF line becomes the new one to be modified.

3. Operation

   (1) When the system is waiting for an editor command, type in the character pattern to be replaced followed by [Y] key and the new character pattern after the prompt mark.

   (2) When you hit the [RETURN] key, the X command is executed from the line to be modified up to the last line. Among the lines searched, only the lines where the replacements have taken place are shown on the screen. If the lines are not displayed in one frame, the system shows the first one frame of the lines. And then if you hit the [SPACE] key, the next one frame of the lines will be displayed. This operation has to bw repeated until the prompt mark appears on the screen.

N.B.1 The character pattern cannot occupy the different fields.

N.B.2 If you want to stop the execution of the Y command, hit the [RESET] switch. Once the [RESET] switch is hit, the system returns to the BASIC mode.

If you want to go back to the Assembler mode, execute the instruction of EXEC&H5006. When this is executed, the system is waiting for an editor command.

Example:

The character pattern to be replaced (ABC → CCC) has been keyed in, and the Y command has been executed.

```
/1
__________ORG___$A0
/YABC/CCC
__CCC_____BNE___XY
__________BRA___CCC
__EOF
/_
 ↑
Blinking Cursor
```

5-13 S Command

1. Formula

S ["] <Device Name> : <File Name> ["] [RETURN]

↑

File Descriptor

2. Function

This command saves the content of the beginning to the last line together with the file name on an output storage device. After this command has been executed, the first line becomes the new one to be modified.

3. Operation

(1) When the system is waiting for an editor command, you can key in the device name and the file name consisting of within 8 characters after [S] ["] keys. If you make a mistake in keying in, you can delete one character by one by using [DEL] key. You can type in new characters.

(2) When the saving on an output storage device is finished, the prompt mark meaning "waiting for an editor command" is displayed.

(3) As for a file name, you can use 8 characters or special symbols in maximum, but the file name cannot contain ":". "0". or "225".

(4) ["] just before the [RETURN] key cannot be omitted, because this editor mode is different from the BASIC mode. The device name and file name have to be enclosed by ["] .

If you omit the last ["] by mistake, the system returns to the BASIC mode. In this case, execute the instruction of EXEC &H5006 to return to this mode.

N.B.1 As for the detailed descriptors, refer to the manuals on Basic Master Level-3 BASIC or Basic Master Level-3 DISK BASIC.

N.B.2 The recording method by this editor command is binary just like that by "SAVEM" in BASIC language binary. When the file is loaded by this command, it is stored at the same location as the saving has taken place. Therefore, you have to remember the top address of your program. You have to designage the address (SOURCE TOP ADDR) when you load the program.

5-14 V Command

1. Formula

   V ["] CASO : <File Name> ["] [RETURN]

2. Function

   The check sum calculated by the program on the cassette tape is compared with the check sum result stored on the cassette tape when the S command is executed. At the completion of this command, the line to be modified moves to the beginning line.

3. Operation

   (1) When the system is waiting for an editor command, key in the device name (CASO:) and the file name followed by [V] ["] .

   (2) Cassette tape has to be re-wound before the program has been recorded.

   (3) The same file name as that recorded by the S command has to be keyed in. When you hit the [RETURN] key, the V command begins to be executed.

   (4) The cassette tape recorder has to be re-wound, and you can re-load your program again.

   (5) If the saving of file is correct, the V command is finished. The system is waiting for another editor command.

   N.B. When Basic Master Level-3 is operating under DISK BASIC, this V command is invalid. This command is applicable only to cassette tape.

5-15 M Command

1. Formula

M ["] <Device Name> : <File Name> ["] [RETURN]
↑
File Descriptor

2. Function

This command merges the source program stored on an external storage device with the line to be modified and the lines thereafter. At the completion of this command the line to be modified moves to the beginning line.

3. Operation

(1) When the system is waiting for an editor command, key in the device name and the file name consisting of within 8 characters, followed by [M] ["] .

(2) If you make a mistake in keying in, you can delete one character by one by using [DEL] key. You can type in new characters.

(3) When the merging is completed, the line to be modified becomes the beginning one and the system is waiting for another command.

4. Others

When you execute the M command, the source top address of your source program stored in the RAM memory has to be the same as that stored on an external storage device.

5-16 A Command

1. Formula

   A [RETURN] or A, [RETURN]

2. Function

   This command converts your source program made by the editor into machine codes, and outputs the assemble listing on the screen. Furthermore, this stores the machine codes in the RAM memory.

   When you use the command of [A] [,] [RETURN] , the system displays one frame of the listing after another.

3. Operation

   (1) When the system is waiting for an editor command, you can key in [A] [RETURN] . Then, the assembling is executed, and and the result is displayed on the screen. The system is waiting for another command.

   (2) When you key in [A] [,] [RETURN] , one frame of the listing is displayed. When you hit the [SPACE] key, the next frame of the listing is displayed on the screen. This operation has to be repeated until the prompt mark appears on the screen.

   N.B. If you terminate the assembling procedure halfway, hit the [RESET] switch. When this is done, the system returns to the BASIC mode. When you want to return to the Assembler mode, execute the instruction of EXEC&H5006. When this executed, the system is waiting for an editor command.

   Example:

   The assemble result is shown in the following format:

```
001  02EF  BD205E
          Machine codes of      If not displayed here,
          1 to 5 bytes          the rest will be shown
                                in the next line
          MEMORY                      MEMORY
LABEL     LOCATION    LABEL           LOCATION
L00P12    &H4012      L00P14          &H40EC

**ERROR COUNT   3  ..... No. of errors
/_
 └ Blinking cursor
```

5-17 AL Command

1. Formula

   AL [RETURN]

2. Function

   This command outputs the assemble listing.

3. Operation

   When the system is waiting for an editor command, you can key in [A] [L] [RETURN] keys. The system executes the assembling of your source program and outputs the result on an printer.

   N.B. The output format by this command is the same as that by the A command.

   N.B. If you stop printing the listing, hit CRTL [D] keys. When you hit the [CTRL] + [D] keys, the system returns to the BASIC mode. If you want to return to the Assembler mode, execute the instruction of EXEC&H5006. When this instruction is executed, the system is waiting for another editor command.

   N.B. The assemble listing is output on a printer having the device of LPT0:.

5-18 N Command

1. Formula

   N [RETURN]

2. Function

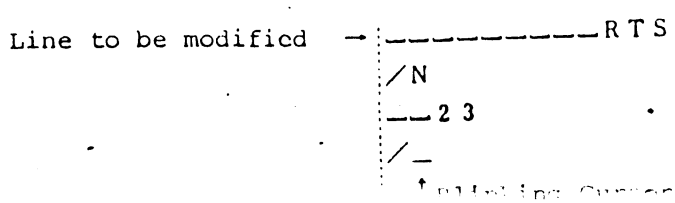
   This command output the line number of the present line to be modified.

3. Operation

   When the system is waiting for an editor command, you can type in [N] [RETURN] keys. The system outputs the line number of the present line to be modified, and then the system is waiting for another editor command.

   Example:

   The system displays the line number of the present line to be modified on the screen.



5-19 C Command

1. Formula

C [ ] [RETURN]

New Prompt Mark

2. Function

(1) This command changes the old prompt mark with the new one designated. This command is valid until the next change of the prompt mark or the re-starting of the editor.

(2) This prompt mark means waiting for an editor command and is used as a space insertion code by the P command. And this prompt mark is used as the demarcation mark between character pattern by X and Y commands.

3. Operation

(1) When the system is waiting for an editor command, you can type in a new prompt mark (only one character) followed by [C] .

(2) When you hit the [RETURN] key, the new prompt mark appears on the screen. The system is waiting for another editor command.

(3) When you have loaded the Assembler, the prompt mark is "/".

Example:

":" has been designated as the new prompt mark by the C command.

```
/C:
:_
```

Blinking Cursor

5-20 S Command

1. Formula

   S [RETURN]

2. Function

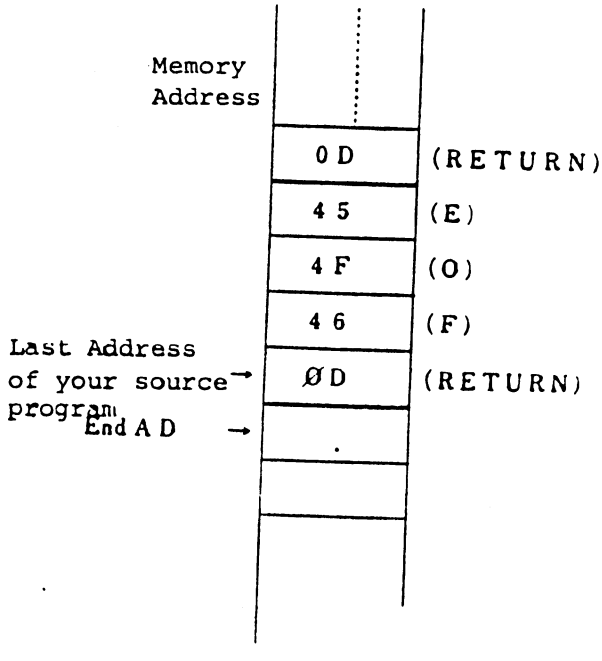
   This command displays the last address of your source program plus 1*.

3. Operation

   When the system is waiting for an editor command, you can key in [S] [RETURN] . The following address format is displayed on the screen.

   End AD = &HXXXX (in Hexadecimal digits)

* The last address of your source program means that EOF is automatically added right after the last line of your source program and that the memory location of [RETURN] for the EOF becomes the last address.



5-21 E Command

1. Formula E

2. Function

This command returns to the BASIC mode.

3. Operation

(1) When the system is waiting for an editor command, you can key in E key. Then the system returns to the BASIC mode and is waiting for the BASIC command.

5-22 Error Message

1. Screen Format

```
/W
**ERROR (with the bell sound initiated)
/-
 t_ Blinking Cursor
```

2. Function

When you have made a mistake in inputting an editor command, the error message will appear on the screen and the bell rings.

3. Operation

You have to input a correct editor command again.

6. ERROR MESSAGE OF ASSEMBLER

During the assembling procedure, error message may appear on the screen, as shown in Table 6-1-1. You have to correct your source program according to the error messages.

Table 6-1-1 Table of Error Messages

ERROR DISPLAY	CONTENT OF ERRORS
***ERROR01	No END in your source program.
***ERROR02	Undefined labels are found. The system insert $0000 value automatically.
***ERROR03	Labels are defined twice. The system treat the first label appearing in your program as valid one.
***ERROR04	The label table has overflowed.
***ERROR05	A label consists of more than 7 characters. The system takes the first 6 characters as legal ones.
***ERROR06	Illegal instructions are found. The system treat the op code and operand as $121212.
***ERROR07	An op code consists of more than 5 characters.
***ERROR08	No register designation in an instruction. Or Illegal registers are used.
***ERROR09	Illegal operand found.
***ERROR10	Relative Address Format is not applicable. The operand value exceeds $FF. The FCB data is of 2 bytes. The displacement exceeds $FF. The calculation result of a symbol is negative.
***ERROR11	Illegal index description. No register designation in an operand. Illegal registers in operands.
***ERROR12	Illegal value found.

***ERROR13	Immediate instructions are not allowed.
***ERROR14	No label to EQU pseudo-executable instruction.
***ERROR15	Illegal usage of Forced Address Format.

N.B. In case of ***ERROR08, the following procedures have been done by the system. "S" register is used for PUL and PSH instructions. "X" register is used for the instructions which accept X, Y, U, and S. "A" register is applied to the instructions which accept the other registers.

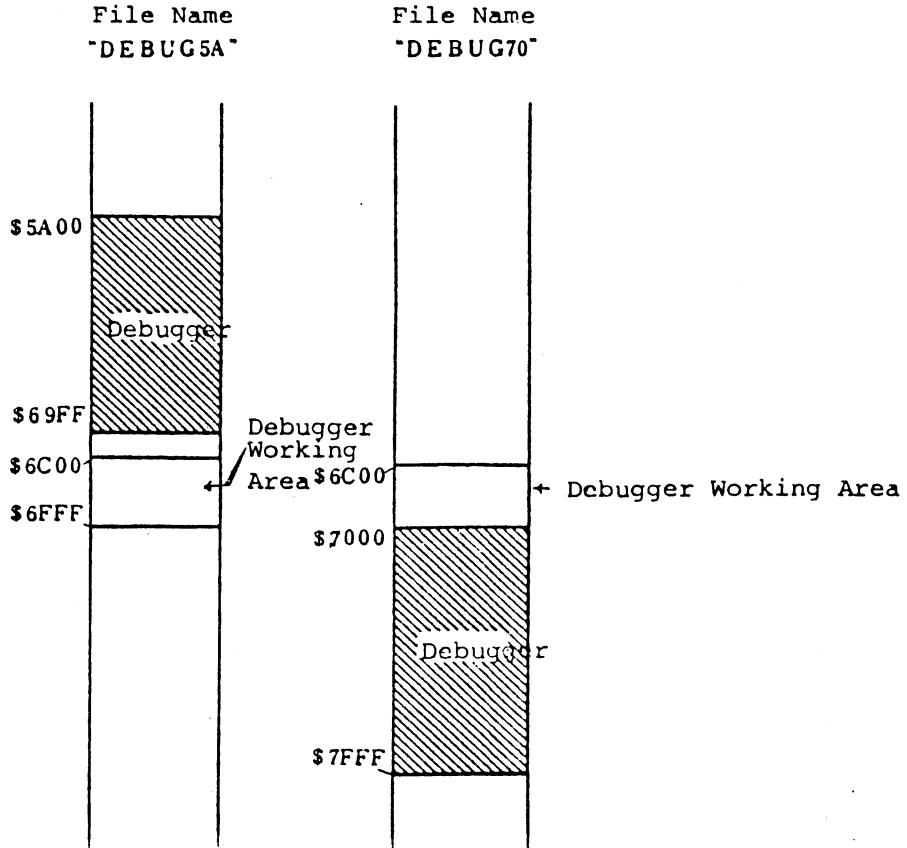
3. DEBUGGER

1. Concept of Debugger

The debugger which has the file name of either DEBUG5A or DEBUG70 is an effective tool to develop machine code programs by Basic Master Level-3.

2. There are DEBUG5A and DEBUG70 Debuggers which use the different storage allocations. The storage and working areas of each Debugger are shown in Fig. 3-2-1. The memory map relationship of these Debuggers with the Assembler and BASIC is described at Section 2-1 in Chapter 1. Refer to this section for details.

Figure 3-2-1 Storage Area of Debuggers



3 Debugger Commands

Effective Debugger commands are available to debug machine codes easily as shown in Figure 3-1-1.

Table 3-1-1 Table of Debugger Commands

CLASSIFICATION	NO	COMMAND	FUNCTION
Memory Reference and Modification	1	M (Memory)	This command refers to and modify the content of the designated memory location. By inputting special symbols, you can move the addresses forwards and backwards.
	2	D (Display)	The contents of 128 bytes (for the 40 character mode) or 256 bytes (for the 80 character mode) are displayed on the screen from the designated address.
Program Execution and Control	3	B (Break Point)	This command sets, releases, and refers to the break point. You can set 5 break points at one time.
	4	G (Go)	The program will be executed from the designated address. When the program has stopped at the designated break point, the contents of all the registers are shown on the screen.
	5	R (Register)	This command refers to and modify the contents of registers. By using special symbols, you can refer to and modify the previous and next registers.
	6	S (Step)	One instruction by one is executed from the designated address (Trace Function). After one instruction has been executed, the contents of all the registers are displayed on the screen.
Memory Block Transfer	7	C (Compare)	The comparison is made between 2 memory areas. If each content at a certain location disagrees, the system displays the date and address of each memory location in each area.

Memory Block Transfer	8	F (Fill)	This command fills some designated data from the designated location to the designated location.
	9	T (Transfer)	This command transfer the contents of some memory area to the designated location and afterwards- If the two memory areas overlap, this command executes the operation.
Others	10	E (Escape)	This command returns the system to the BASIC mode.
	11	A (Assign)	As an output device this command designates either of Display, Printer, or Display and Printer.

Other than the Debugger commands shown in Table 3-1-1, the special keys shown below have the following functions:

(1) When you type in the [SPACE] key, the previously keyed-in command will be executed.

(2) When you type in the [RETURN] key, the system is always waiting for a Debugger command.

(3) When you type in [DEL] key, the previously input data will be neglected. And the system is waiting for another data input.

(4) When an error has taken place, the bell rings and a question mark (?) is displayed on the screen.

3-1 M Command

1. Function

(1) This command displays the content of a designated memory location.

(2) This command modifies the content at a memory location and writes with the new contents.

2. Operation

(1) When the system is waiting for a Debugger command, you can type in the [M] command and the system is waiting for an address input.

(2) When you input an address and then the [SPACE] key, the present content of the memory location is displayed on the screen. The system is waiting for data input. As shown below, the several functions are available by using special keys.

NO	INPUT KEYS	FUNCTION
1	[SPACE]	This displays the content at the next memory location.
2	[/]	This displays the content at the previous memory location.
3	[RETURN]	The system is waiting for a Debugger command.
4	[^]	The system is waiting for an address input shown No. 1.
5	<Data>* Any key from No. 1 to No. 4	This writes new data at designated memory location.

N.B.1 The operations from No. 1 to No. 4 do not modify the content of a memory location.

N.B.2 When you want to access the memory location where no RAM is mounted or when you modify the content of the ROM, no data is written and the question mark (?) is displayed together with the bell sound.

Example: The underlined part has to input.

Address Data

COMMAND : [M]

ADDRESS : ××××-××××[SPAC

3-2 D Command

1. Function

(1) The memory contents from the designated locations thereafter are displayed on the screen.

SCREEN FORMAT	NO. OF BYTES SHOWN
40 Character mode	128 Bytes
80 Character mode	256 Bytes

(2) The characters are displayed on the screen according to the the contents of the locations.

2. Operation

(1) When the system is waiting for a Debugger command, you can key in [D] key. And the D command has been executed and the system is waiting for address input.

(2) When you type in the beginning address where the memory contents are displayed and when you type in [SPACE] key, the system shows 8 bytes in each of 16 lines*. In case of the 80 character mode, the system displays 16 byte contents in each of the 16 lines**. At the end of the command, the beginning address of the next memory block is displayed on the screen.

(3) When you want to continue this operation, just type in [SPACE] key. If you want to change the address, just type in the address and then SPACE key.

(4) When you want to terminate the command, type in [RETURN] key.

N.B.1* This is applied to the 40 character mode only.
** This is only applied to the 80 character mode.

N.B.2 When you hit [^] or [/] key instead of the [SPACE] key, the previous 128 bytes (256 bytes for the 80 character mode) are shown on the screen.

Example: The underlines part has to be input.

COMMAND : [D]
ADDRESS : ××××␣-␣××××[SPACE]

↑ Address ↑ Address

3-3 B Command

1. Function

(1) This command sets, displays, modifies, and releases a break point*.

(2) Five break points can be set at one time.

2. Operation

(1) When the system is waiting for a Debugger command and when you type in B key, the system displays the five break points which have been currently set. And the system is waiting for the first input of the break point.

(2) While the system is waiting for the address input, the following operations are possible by using the special keys shown below.

NO	INPUT KEYS	FUNCTION
1	[SPACE]	The system shows the next break point.
2	[/] or [^]	The system shows the previous break point.
3	[RETURN]	The system is waiting for another Debugger command.
4	<Address> [SPACE]	The system sets or modified a break point.
5	0 (Zero) [SPACE]	The system releases the break point already set.

* A break point in machine language works just line the STOP statement in BASIC language. You can set these break points at any locations in order to debug your program under development. The main purpose is to check the contents of registers.

N.B. When you set a break point inside one instruction, the system never stops there.

Example:

The underlined part has to be input.

```
COMMAND: B
BREAK POINT
  1       2       3       4       5
××××  ××××  ××××  ××××  ××××
1 : ××××  - ××××  [SPACE]
2 : ××××  - ××××  [SPACE]
5 : ××××  - ××××  [SPACE]
```

3-4 G Command

1. Function

(1) This command executes your program from the designated address by the G command.

(2) When the break point has already been set, the program stops at this point and contents of all the registers are displayed.

2. Operation

(1) When the system is waiting for a Debugger command and you type in [G] key, the G command is initiated. The system shows the address of the program counter and is waiting for the change of the content of the program counter.

(2) When you want to execute your program from the displayed address, just type in [SPACE] key. When you want to change the execution address, key in the new address and hit the [SPACE] key.

N.B.1 This G command does not execute the Assembler.

N.B.2 In the following are shown the cases where your program execution is stopped:

(1) In case where some break points have been set in your program, the G command executes up to the previous instruction before the break point already set. All the register contents are displayed.

(2) When you hit the [BREAK] key during the program execution, your program execution is stopped and all the register contents are displayed.

N.B.3 The program which has been stopped at a break point is re-started by keying in the [SPACE] key until the next break point is found. As shown below, the break points set by the B command can be skipped and you can designate the number of skipping the break points already set.

```
.....USP   PC
     ××××␣××××␣3 [SPACE]
                  └ keyed in
```

In this case, 2 break points will be skipped from the present break point and the program will be executed until the third break point point. The number you can designate is &H01 to &HFF. But this Debugger only accepts 5 break points. Therefore, if you designate more than 5, all the break points already set will be skipped.

Example:

The underlined part has to be input.

COMMAND : G

ADDRESS : ××××␣-␣×××× SPACE

Address Address

3-5 R Command

1. Function

This command displays and modifies the contents of the registers.

2. Operation

(1) When the system is waiting for a Debugger command and you have keyed in R key, the R command is executed and the register contents are displayed.

(2) At the completion of displaying all the register contents, the system is waiting for the inout to the SP register.

The following operations are possible:

NO	INPUT KEYS	FUNCTION
1	[SPACE]	The next register content is shown.
2	[/] or [^]	The previous register content is shown.
3	[RETURN]	The system is waiting for another Debugger command.
4	<Data*> [SPACE]	The system writes data to the displayed register.

* 1 byte or 2 bytes of data

Example:

The underlined part has to be input.

```
COMMAND: [R]

  SP   CC   A   B DP   IX    IY   USP   PC
××××␣××␣××␣××␣××␣××××␣××××␣××××␣××××
SP␣××××␣-␣×××× [SPACE]
CC␣××␣-␣×× [SPACE]
␣A␣××␣-␣×× [SPACE]
␣B␣××␣-␣×× [SPACE]
DP␣××␣-␣×× [SPACE]
␣X␣××××␣-␣×××× [SPACE]
␣Y␣××××␣-␣×××× [SPACE]
␣U␣××××␣-␣×××× [SPACE]
PC␣××××␣-␣×××× [SPACE]
```

3-6 S Command

1. Function

(1) This command traces one instruction by one from the address of the program counter.

(2) After the execution of one instruction, all the register contents are displayed.

(3) Even if a break point has been set, this command executes one instruction by one.

2. Operation

(1) Before the S command, you have to change the content of the program counter to the beginning address from which you want to execute the S command.

(2) When the system is waiting for a Debugger command and you type in the S eky, the S command is initiated and the system is waiting for the data input.

(3) When you key in [SPACE] key, the system shows the address of the program counter shown in (2).

(4) When you hit the [RETURN] key, the system is waiting for another Debugger command.

Example:

The underlined part has to be input.

```
COMMAND : S
                                  ┌─ This is shown in case of 2 bytes
STEP␣××××␣-␣××(␣××)pf an instruction
  SP   CC   A   B  DP   IX    IY   USP   PC
××××␣××␣××␣××␣××␣××××␣××××␣××××␣××××␣SPACE
```

N.B When you key in the SPACE key one by one, the program is executed one step by one, but you can skip the program execution designated times as shown below:

```
USP   PC
××××␣××××␣FF SPACE
                └─ Keyed in
```

In the above case, the system skips the program execution 254 steps, and the 255th instruction is executed. The number you can designate is &H01 to &HFF.

3-7 C Command

1. Function

This command compares the contents of 2 memory blocks.

2. Operation

(1) When the system is waiting for a Debugger command and you type in the [C] key, the C command is initiated and the system is waiting for the input of the beginning address of the master memory block.

(2) When you key in the address and the [SPACE] key, the system is waiting for the end address of the master memory block.

(3) When you key in the address and hit the [SPACE] key, the system is waiting for the input of the beginning address of the memory block to be compared.

(4) When you key in the address and hit the [SPACE] key, the execution of the S command starts. If the contents of the 2 memory blocks agree, the system is waiting for another Debugger command.

(5) If the contents disagree, the system displays the addresses and data of each block and is waiting for another data input.

(6) If you hit the [SPACE] key under the condition of (5), the system shows the address and data afterwards when the disparity of data has taken place. Then, the system is waiting for another data input.

(7) When you hit the [RETURN] key, the command will be terminated.

Example:

The underlined part has to be input.

```
COMMAND:C
MASTER
FROM␣××××␣-␣××××SPACE
␣TO␣␣××××␣-␣××××SPACE
DESTIN.
FROM␣××××␣-␣××××SPACE
××××␣-␣××␣:××××␣-␣××␣?
Address   Data   Address   Data
```

Display format when the disparity has taken place

Blinking Cursor

Master Memory Block

Memory Block to be compared

3-8 F Command

1. Function

This command fills a constant at the designated locations. When the constant is not filled correctly*, the system displays the address and the content.

2. Operation

(1) When the system is waiting for a Debugger command and you type in [F] key, the F command is initiated. The system is waiting for the input data to be filled.

(2) When you hit the [RETURN] key, the system is waiting for the beginning address of the memory block where you want to input new data.

(3) When you type in the end address and hit the [SPACE] key, the F command is executed. The system is waiting for another Debugger command.

* The case where data has not been filled correctly means that you have designated the address of no RAM or that you have tries to write data on ROM.

N.B. If you wirte data in the working area of the system, the system goes crazy.

Example:

The underlined part has to be input.

COMMAND : [F]
DATA␣××␣-␣<u>×× [SPACE]</u>
FROM␣××××␣-␣<u>×××× [SPACE]</u>
␣TO␣␣××××␣-␣<u>×××× [SPACE]</u>

3-9 T Command

1. Function

(1) This command transfers data from designated memory area to the other area.

(2) Even if the memory addresses overlap, this command is executed.

(3) If the memory transfer is not executed, the system displays the addresses and data at the original and transferred locations where the disparity has taken place*.

2. Operation

(1) When the system is waiting for a Debugger command you type in the T command, the [T] command is initiated. The system is waiting for the beginning address of the original memory block.

(2) When you type in the address and hit the [SPACE] key, the system is waiting for the end address of the original memory block.

(3) When you key in the address and hit the [SPACE] key, the system is waiting for the input of the beginning address where data is transferred.

(4) When you key in the address and hit the [SPACE] key, the transfer is executed. At the completion of the memory transfer, the system is waiting for another Debugger command.

* The case where the correct transfer has not taken place means that you have designated the address in no RAM or that you have tries to transfer to the ROM area.

N.B. If you transfer to the working area of the system, the system goes crazy.

Example:

The underlined part has to be input.

```
COMMAND : [T]
MASTER
FROM␣××××␣-␣×××× [SPACE]
␣TO␣␣××××␣-␣×××× [SPACE]
DESTIN.
FROM␣××××␣-␣×××× [SPACE]
```

3-10 E Command

1. Function

This command returns the system to the BASIC mode.

2. Operation

When the system is waiting for a Debugger command and type in the E key, this command returns the control to the BASIC mode.

3-11 A Command

1. Function

As an output device, the command designates one of the following devices:

(1) Display (CRT)
(2) Printer (LPT)
(3) Display and Printer (CRT + LPT)

2. Operation

(1) When the system is waiting for a Debugger command and you type in the A key, the A command is initiated and the system is waiting for the designation of an output device.

(2) When you type in the number of 1 to 3 which designates an output devices, the designated device becomes valid afterwards.

(3) If you want to change the designation of an output device, this command has to be executed again.

N.B. When you designate Display or Display and Printer, make sure that a printer has been connected to the system.

N.B. When you designate a printer, the content that has been typed in previously will not be printed until you hit the SPACE key.

N.B. If you do not designate the output device by the A command, the system automatically designate CRT by default.

N.B. The device name of a printer designated as an output device is LPT0:

Example:

The underlined part has to be input.

COMMAND : [A]
CRT=1,LPT=2,CRT+LPT=3 : [1] or [2] or [3]

4. OPERATION OF BASIC MASTER LEVEL-3

1. Concept

The Assembler is available either on cassette (MA-5100) or on diskette (MA-5200). The application of the Assembler is different according to the memory size of Basic Master Level-3. At the following sections we are going to describe the details of the Assembler.

It is recommended to use the 80 character mode for the screen of MB-6890, which makes it easy to read the assembled list on the screen.

2. Operation of MA-5100 (Cassette Tape)

2-1 Re-Saving of the Assembler

The Assembler is recorded on cassette tape under the file name of "ASM9" as shown in figure 4-2-1. The Debuggers are recorded under the file names of "DEBUG70" and "DEBUG5A". DEBUG70 is stored at the memory locations of &H7000 to &H7FFF. DEBUG5A is stored at the memory locations of &H5A00 to &H69FF. For cassette-based system, DEBUG70 has to be used.

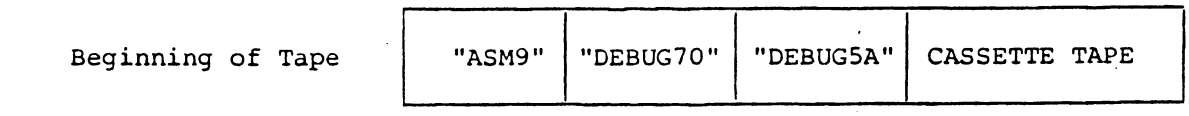


Figure 4-2-1 File Organisation of Cassette Tape

According to the following procedures, these programs are re-recorded on another cassette tape and it is recommended to use the re-recorded tape. The original one should be kept as master tape.

(1) A cassette tape recorder should be connected to Basic Master Level-3 and the cassette tape, MA-5100, should be installed. You can load the program under the file name of "ASM9" by using the BASIC command.

Example:

```
Ready
LOADM "ASM9"  [RETURN]

Searching          ← The tape recorder has to be under playing
                     position
Found: ASM9
Ready
_
↳ Blinking Cursor
```

The Debugger can be loaded under the file name of "DEBUG70".

(2) Another cassette tape has to be installed in the cassette player, and you can save the program on this tape by using the BASIC command.

Example:

```
Ready
SAVEM "ASM9", &H5000, &H69FF &H5000 [RETURN]
Ready
```

N.B. Before you hit the RETURN key, you have to push the "Record" button and start the recorder.

When the saving of the Assembler is finished, give the following command:

```
SAVEM "DEBUG70", &H7000, &H7FFF, &H7000 [RETURN]
```

This command saves the Debugger on the tape.

From now on, you can use re-saved tape.

2-2 Loading of Assembler and Debugger

You can load the program of the Assembler and Debugger in Basic Master Level-3 from the re-saved tape.

(1) Install the tape mentioned before in a tape recorder.

(2) Reserve the memory area for the machine codes by using the BASIC command.

Example:

```
CLEAR300, &H4EFF [RETURN]
          └ &H4EFF Do not use the address greater than &H4EFF
```

N.B. When you expand the memory area up to 40KB and use the MA-5100, There is no need to reserve the memory area of the object program by using the BASIC command. When you use the CLEAR command, refer to Section 2-1 in Chapter 1 (Memory Map).

(3) The Assembler will be loaded by the LOADM in the BASIC language.

Example:

```
Ready
LOADM "ASM9" [RETURN]

Searching                    ← The tape recorder has to be set
                               to the playing position
Found: ASM9
Ready
_
↑— Blinking Cursor
```

(4) You can load the Debugger under the file name of "DEBUG70" in the same way as the Assembler. If you do not want to use the Debugger, there is no need to load it. If you do not use the Debugger, the area allocated for the Debugger can be used for that for your source program or its object program.

N.B. When you switch off the machine, the Assembler or Debugger disappears. Every time when you switch on the machine, you have to load them.

N.B. When the object program is destroyed and you run the program, the Assembler or Debugger will be destroyed. In this case, re-load them again.

2-3 Execution of Assembler

(1) You can start the assembler from the beginning address by the EXEC command in the BASIC language as shown in Table 4-2-1.

Table 4-2-1

EXECUTION METHOD	STARTING ADDRESS
NORMAL START	&H5000
1. Re-start after the E command in the Editor 2. Re-Start after the [RESET] 3. Re-Start after [CTRL] + [D]	&H5006
LOADM"ASM9",,R [RETURN]	(&H5000)

In case of the re-starting the Assembler, the system is waiting for an editor command. In case of LOADM"ASM9",, R, the Assembler is initiated. This is the same with the normal start. The normal start is shown in the following example:

Example:

```
Ready
EXEC&H5000 [RETURN]

* ASMB/EDITR-V2.0
:NEW/RAM/EXTM ? _
                └ Blinking Cursor
```

N.B. By using the G command in the Monitor, the Assembler can be initiated. But by using the same command, you can not initiate the Debugger.

(2) Input Deisgnation of Source Program

As input methods of your source program, there are 3 kinds. By by inputting one character as shown below, you can choose one of of them.

(1) The case where you input your source program through the keyboard:

```
:NEW/RAM/EXTM ?[N]
/SOURCE TOP ADDR ?
                  └ Blinking Cursor
```

(2) The case where your source program is already in the RAM:

```
:NEW/RAM/EXTM ?[R]
/
└ Blinking Cursor
```

(3) The case where your source program is loaded from the cassette tape:

```
:NEW/RAM/EXTM ?[E]
/SOURCE TOP ADDR ? _
(Input the top address          └ Blinking Cursor
 of your source program)
PROGRAM NAME : EXAMPLE" [RETURN]
(Program name to be                └ Input the file name of within
 loaded)                             8 characters, and start the tape
/                                    recorder
└ Blinking Cursor
```

*PROGRAM NAME : CAS0 : EXAMPLE [RETURN] (You can designate the device name as (*).

(3) Designation of the Memory Address of Source Program

In reference to the Memory Map describes at Section 2-1 in Chapter 1, you can decide on the starting address of your source program in the RAM and input the address in Hex.

If you input more than 4 digist, the last 4 digits will be accepted. When you input the memory location lower than the beginning address of the user's RAM, the system is waiting for another address input.

Example:

```
*ASMB/EDITR-V2.0
:NEW/RAM/EXTM?N
/SOURCE TOP ADDR?3000 RETURN
/
 L  Blinking Cursor
```

N.B.1 If you designate the top address of the user's RAM as the beginning address of your source program and execute the BASIC commands under the BASIC mode, there is possibility that your source program will be destroyed. In this case, it is recommended to designate the beginning address as user's beginning address of the RAM plus &H100.

N.B.2 When you load your source program saved on cassette, you have to designate the same address as you have saved it on tape. If both addresses are different, the program will not be loaded correctly.

2-4 Execution of Assembler

(1) When you have finished the editing of your source program by the Editor, type in the A command.

Example:

/A RETURN

(The assembled list will be displayed on the screen)

/A, RETURN

(2) If the listing exceeds one frame of the screen, the system shows the first frame of the listing. After checking the list, type in the RETURN key.

When the number of errors and the prompt mark appear on the screen, that is the end of the listing.

The object program is stored in the RAM according to the ORG instruction. Therefore, you can execute the program right away.

If some errors are found, you can modify your program under the Editor.

(3) If you want to assemble your source program saved on cassette tape, you have to follows the same procedures mentioned above.

(4) If you want to output the assemble listing on a printer, you have to connect the printer to the system and give the AL command.

Example:

/AL [RETURN]

If you want to stop printing halfway, just type in [CRT] + [D] keys. If you hit [CTRL] + [D] keys, the system returns to the BASIC mode. To return to the Assembler mode, execute the command of EXEC&H5006 in the BASIC language. The system will return to the Assembler mode and is waiting for an editor command.

N.B. The assembled listing is output on a printer which has the device name of LPTP:

(5) The errors which have happened during the assembling procedures are represented by the numbers shown in Table 2-6-1 in Chapter 2. And the assembling procedures will be continued under the conditions shown there.

(6) To execute your object program, you have to execute an instruction of EXEC after you have returned to the BASIC mode by the E command in the Editor.

Example:

/E EXEC&HXXXX [RETURN]

Or you can execute your object program by the G command in the Monitor.

Example: (After you have returned to Monitor mode)

*GXXXX [RETURN]

N.B. The memory locations of &H400 to &H43FF are used for a colour display unit. If you use this memory area for the storage of your object program or the stack, the colour of a monitor will be affected. When you use this memory area for the storage of your object program, you have to bare this fact in mind.

2-5 Operation of Debugger

(1) Start from the address of &H7000 by the instruction of EXEC in the BASIC language.

Example:
```
Ready
EXEC&H7000 [RETURN]
(The screen will be cleared)
**  DEBUGGER  V1.0  **
COMMAND:                    Waiting for a Debugger command
```

(2) The Debugger will be initiated by the G command in the Monitor.

Example: (After you are in the Monitor mode)

*G7000 | RETURN |

(3) You can also initiate by LOADM,,R command in the BASIC just like the Assembler.

Example:

LOADM "DEBUG70",,R | RETURN |

In this case the Debugger will be initiated after the DEBUG70 has been loaded.

3. OPERATION OF MA-5200 (DISKETTE)

3-1 Operation of MA-5300 (32K DISK BASIC)

3-1-1 Resaving of Diskette

On the diskette MA-5200, the Assembler is saved under the file name of ASM9 and the Debugger is saved under the file names of DEBUG5A and DEBUG70. The DEBUG5A is loaded in &H5A00 to &H69FF in the RAM and the DEBUG70 is loaded at &H7000 to &H7FFF. The DEBUG70 overlaps the memory area with 32KB DISK BASIC. Therefore, this cannot be used. You have to use DEBUG5A. According to the following procedures, it is recommended to resave the programs on another diskette and to use it all the time. You should keep the original diskette as a master program file.

(1) After you have initiated the DISK BASIC (MA-5300), insert the diskette, MA-5200, into the Drive 1 of the mini floppy disk unit and another blank one into the Drive 0. By using the BASIC command, load the Assembler under the file name of ASM9.

Example:

```
Ready
LOADM "1:ASM9" [RETURN]
Ready

└ Blinking Cursor
```

N.B. In order not to destory MA-5200 by mistake, make sure that the write protection seal is stuck on the diskette.

(2) Then save the Assembler on the diskette in the Drive 0.

```
Ready
SAVEM "0:ASM9", &H5000, &H69FF, &H5000 [RETURN]
Ready
```

N.B. The write protection seal on the diskette in the Drive 0 should be taken out beforehand.

(3) After the above-mentioned operation, load DEBUG5A from the MA-5200.

Example:

```
Ready
LOADM "1:DEBUG5A" [RETURN]
Ready

└ Blinking Cursor
```

(4) Save the program on the diskette in the Drive 0.

Example:

R e a d y

S A V E M "0 : DEBUG5A", &H5A00, &H69FF, &HFA00 [RETURN]

After the saving, stick the write protection seal.

3-1-2 Loading of Assembler

Load the Assembler into Basic Master Level-3 from the resaved diskette.

(1) Initiate 32KB DISK BASIC.

(2) Insert the resaved diskette into the mini floppy disk unit.

(3) You have to reserve the memory area for the object program by using the CLEAR command in the BASIC language.

Example:

CLEAR 300, &H4EFF [RETURN]

↑

The address should not be greater than &H4EFF.

(4) The Assembler is loaded by the LOADM command in the BASIC.

Example:

R e a d y

L O A D M "1 : ASM9" [RETURN] This is applicable when the diskette in the Drive 1

R e a d y

-

↑ Blinking Cursor

3-1-3 Operation of Assembler

(1) You have to start from the address shown in Table 4-3-1 by the EXEC command in the BASIC.

Table 4-3-1 Beginning Address of Assembler

STARTING METHOD	STARTING ADDRESS
NORMAL START	&H5000
1. Re-start after the E command in the Editor	&H5006
2. Re-start after the [RESET]	
3. Re-start after [CTRL] + [D]	
LOADM"DEVICE NAME* : ASM9",,R [RETURN]	(&H5000)

* As for the device names, refer to the manuals on BASIC Master Level-3 ROM BASIC or DISK BASIC.

In case of the re-start mentioned above, the system is waiting for an editor command. In case of using LOADM,,R command, the Assembler is initiated after the loading, which is exactly the same with the normal start. The normal start is shown in the following:

Example:

```
R e a d y
EXEC&H5000 [RETURN]
(The screen will be cleared)
* ASMB/EDITR  V2.0
:NEW/RAM/EXTM ?
```

t Blinking Cursor

N.B. You can initiate the Assembler from the start address shown in Table 4-3-1 by using the G command in the Monitor. But you cannot initiate the Assembler by the G command in the Debugger.

(2) Input Designation of Source Program

As input methods of your source programs, there are 3 ways. You can designate one of them by inputting one character in the following:

(1) The case where your source program is input through the keyboard:

```
:NEW/RAM/EXTM? [N]
/SOURCE TOP ADDR?
```

(2) The case where the program is in the RAM:

```
:NEW/RAM/EXTM? R
/ _
  ↑ Blinking Cursor
```

(3) The case where the program is loaded from the diskette:

```
:NEW/RAM/EXTM?  E
/SOURCE TOP ADDR ? _
(Input the top      ↑ Blinking Cursor
address of your
source program)
PROGRAM NAME:1:EXAMPLE [RETURN]  (This is the case where the
(Program Name ↑ Device Name and    diskette is inserted into Drive 1)
to be loaded)   File Name of within
                8 Characters
/ _
  ↑ Blinking Cursor
```

N.B. There are no need of the double quotation marks before and after the device name and file name.

(4) The case where the program is loaded from the cassette tape:

```
:NEW/RAM/EXTM?  E
/SOURCE TOP ADDR? _
                  ↑ Blinking Cursor

(Input the top address of your source program)
PROGRAM NAME:CASO:EXAMPLE [RETURN]
             ↑ Device name CASO: of the cassette tape
               and the file name of within 8 characters
               have to be input, and start the tape recorder.
(Program to be loaded)
/ _
  ↑ Blinking Cursor
```

N.B. The double quotation marks are not needed before and aftter the device name and file name of within 8 characters.

(3) Designation of Beginning Address of Source Program RAM

In reference to the Memory Map at Section 2-1 in Chapter 1, you have to decide on the beginning address of your source program in the RAM area, and to input the address in Hex. If you input more than 4 digits, the last 4 digits will be accepted. And if you input the address lower than the beginning address of user's RAM, the system is waiting for another input.

Example:

```
*ASMB/EDITR__V2.0__
:NEW/RAM/EXTM? _N_
/SOURCE TOP ADDR? 3000 [RETURN]
/ _
```

N.B. When you designate the beginning address of user's RAM as the top address of your source program and execute the commands in the BASIC language, there is possibility that your source program might be destroyed. It is recommended to designate the address as the beginning address of user's RAM plus &H100 or thereafter.

N.B. When you load programs from the diskette, you have to use the same address as it has been saved. If the addresses disagree, the program will not be loaded correctly. This is the same with the cassette tape.

3-1-4 Execution of Assembler

(1) When you have finished the editing of your source program by editor commands, input the A command.

Example: /A [RETURN]

(The assembled listing is displayed on the screen)

or

/A, [RETURN]

(2) When you type in A, [RETURN] , only one frame of the listing is displayed on the screen. After checking the list, just type in [RETURN] key. When the number of errors and the prompt mark are displayed on the screen, the listing has been completed. The object program is stored in the RAM at the designated locations by the ORG instruction. You can execute your program right away. If you have found some errors, you can modify immediately by the editor commands.

(3) If you want to assemble your source program on the diskette, you can execute the A command in the same way as mentioned before.

(4) If you want to print the assemble listing on a printer, use the AL command. Make sure that the printer is connected to the machine.

Example: /AL [RETURN]

If you want to stop printing halfway, give the command [CTRL] + [D] . When you type in [CTRL] + [D] , the system returns to the BASIC mode. If you give the command of EXEC&H5006, the system returns to the Assembler mode. And the system is waiting for an editor command.

N.B. The listing will be printed on the printer with the device name of LPT0:.

(5) When errors have taken place during the assembling procedures, the type of an error is shown in number shown in Table 2-6-1. The assembling will be continued under the conditions shown there.

(6) To execute your object program, return to the BASIC mode by the E command in the BASIC and execute the instruction of the EXEC command.

Example: /E
EXEC&HXXXX [RETURN]

OR you can execute the object program by the G command in the Monitor.

Example: (In the Monitor mode)
*GXXXX [RETURN]

N.B. The program for the colour display monitor is stored at the memory locations of &H400 to &H43FF. If you use these memory area for the storage of your object program or for the stack operation, the colour of the monitor will be affected. You have to bear this fact in mind.

3-1-5 Loading of Debugger (DEBUG5A)

The purpose of this Debugger is to debug the machine codes of your source program made from your source program. When you debug the object program on the machine code level, you have to load DEBUG5A from the resaved diskette as mentioned in 3-1-1.

(1) There is no need to execute the CLEAR command in the BASIC, because the storage area for the object program is already reserved. Only when you want to use the Assembler, you have to set the storage area of the object program by using the CLEAR command in the BASIC. See the example below:

Example: CLEAR300, &H59FF [RETURN]
↑ Do not designate the address greater than &H59FF.

(2) The Debugger is loaded by the LOADM command in the BASIC.

Example:
```
R e a d y
L O A D M"1:DEBUG5A" [RETURN]   This is the case where
                               the resaved diskette
R e a d y                      is inserted in the Drive 1
  -
  ↑ Blinking Cursor
```

Because the storage area for the Assembler is from &H5000 to &H6FFF and that for the Debugger is from &H5A00 to &H69FF, the Assembler will be destroyed if the Debugger loaded. But the source program and the object program will not be destroyed. Therefore, if you reload the Assembler, you can modify your source program by using editor commands.

N.B. DEBUG70 cannot by used because the storage area for 32KB DISK BASIC overlaps with that for the Assembler.

3-1-6 Operation of Debugger (DEBUG5A)

(1) The Debugger which has been loaded in the above method can be started from &H5A00 by using the EXEC command in the BASIC.

Example:
```
R e a d y
EXEC&H5A00 [RETURN]
(The screen will be cleared)
** DEBUGGER V1.0 **
COMMAND:_
        t Blinking Cursor
```

(2) This can be operated by the G command in the Monitor

Example: (In the Monitor mode)

```
*G5A00 [RETURN]
```

(3) The LOADM,,R command in the BASIC can initiate the Debugger as in the Assembler.

Example: LOADM"DEBUG5A",,R [RETURN]

In this case the Debugger will be initiated right after DEBUG5A has been loaded.

3-2 Operation of MA-5301 (40KB DISK BASIC)

3-2-1 Resaving of the Program

On diskette MA-5200, the Assembler is saved under the file name of ASM9 and the Debuggers are under the file names of DEBUG5A and DEBUG70. DEBUG5A is stored at the memory locations of &H5A00 to &H69FF and DEBUG70 is stored at the locations of &H7000 to &H7FFF. You have to use the DEBUG70 here. Normally, you have to resave these programs on another diskette according to the following procedures. It is recommended to use the resaved one. The original one should be kept as a master diskette.

(1) After the DISK BASIC (MA-5301) has been loaded, insert the diskette, MA-5200, into the Drive 1 and another diskette in the Drive 0. Load the Assembler under the file name of ASM9 by using the BASIC command.

Example:
```
R e a d y
L O A D M"1:ASM9" [RETURN]
R e a d y
-
  t Blinking Cursor
```

N.B. In order not to destroy the diskette, MA-5200, make sure that the protection seal is stuck on the diskette.

(2) Load the Debugger under the file name of DEBUG70 in the same way as mentioned above.

(3) Resave these programs on another diskette inserted into the Drive 0.

Example:
```
R e a d y
S A V E M"0:DEBUG70", &H7000, &H7FFF, &H7000 [RETURN]
R e a d y
-
t Blinking Cursor
```

N.B. After saving these programs, stick the write protection seal on the diskette.

3-2-2 Loading of Assembler and Debugger

Load the Assembler and Debugger into Basic Master Level-3 from the diskette saved in the above-mentioned method.

(1) Load 40KB DISK BASIC (MA-5301)

(2) The resaved diskette has to be inserted into the Drive.

(3) The Assembler is loaded by the LOADM command in the BASIC.

Example:
```
R e a d y
L O A D M"1:ASM9" [RETURN]
R e a d y
-
t Blinking Cursor
```
This is the case where the diskette is inserted into the Drive 1.

(4) The Debugger under the file name of DEBUG70 is loaded in the same way as in the Assembler. If you do not want to use the Debugger, there is no need to load it. If you do not load the Debugger, the area for the Debugger can be used for the storage area for your source program or object program.

N.B. In case of using MA-5301 (40KB DISK BASIC), there is no need to reserve the memory area for the object program by the CLEAR command in the BASIC in order to store the Assembler or Debugger. In order to execute the CLEAR command, refer to Memory Map at Section 2-1 Chapter 1.

3-2-3 Operation of Assembler

(1) Start the Assembler from the address shown in Table 4-3-2 by using the EXEC command in the BASIC.

Table 4-3-2

STARTING METHOD	STARTING ADDRESS
NORMAL START	&H5000
1. Re-start after the E command execution in the Editor. 2. Re-start after the [RESET]. 3. Re-start after [CTRL] + [D]	&H5006
LOADM"DEVICE NAME*:ASM9",,R [RETURN]	(&H5000)

* As for the device names refer to the manuals on Basic Master Level-3 ROM BASIC or DISK BASIC.

In case of the re-start, the system is waiting for an editor command. When you use LOADM,,R command, the system is in the Assembler mode. This is the same with the normal start. The example of the normal start is shown in the following:

Example: R e a d y
EXEC&H5000 [RETURN]
(The screen will be cleared)
* ASMB/EDITR_V2.o
:NEW/RAM/EXTM? _
↑ Blinking Cursor

N.B. By using the G command in the Monitor, the Assembler will be initiated from the starting address shown in Table 4-3-2. This is not applied by the G command in the Debugger.

(2) Input Designation of Source Program

There are 3 methods to input your source program in the RAM. By inputting one character shown below, you can designate the input of your source program.

(1) The case where your source program is input through the keyboard:

:NEW/RAM/EXTM? [N]
/SOURCE TOP ADDR? _
↑ Blinking Cursor

(2) The case where your source program is in the RAM:

```
:NEW/RAM/EXTM? [ R ]
/ _
  t Blinking Cursor
```

(3) The case where your source program is on the Diskette:

```
:NEW/RAM/EXTM? [ E ]
/SOURCE TOP ADDR? _
(Input the top                t Blinking Cursor
address of your
source program)

PROGRAM NAME:1:EXAMPLE [RETURN]  This is the case where the
             t The device name  diskette is inserted into
               and the file name the Drive 1.
               of within 8 characters.
Program to be loaded
/ _
  t Blinking Cursor
```

N.B. The double quotation marks are not needed before nad after the device name and the file name of within 8 characters.

(4) The case where the program is stored on cassette tape:

```
:NEW/RAM/EXTM? [ E ]
/SOURCE TOP ADDR? _
                  t Blinking Cursor

(Input the top address of your source program)

PROGRAM NAME:CASO:EXAMPLE [ RETURN ]
             t Input the Device name CASO: of the
               cassette and the file name of the
               cassette and start the tape recorder.
(Program to be loaded)
/ _
  t Blinking Cursor
```

N.B. The double quotation marks are not needed before and after the device name and the file name of within 8 characters.

(3) Designation of Beginning Address of Source Program

In reference to Memory Map at Section 2-1 in Chapter 1, you have to decide on the beginning address of your source program in the RAM, and to input the address in Hex. If you input more than 4 digits, the last 4 digits will be effective. And if you designate the address lower than the beginning address of user's RAM, the system is waiting for another input.

```
Example:  *ASMB/EDITR__V2.0
          :NEW/RAM/EXTM? [ N ]
          /SOURCE TOP ADDR?  3000 [ RETURN ]
          / _
            t_ Blinking Cursor
```

N.B.1 When you designate the beginning address of user's RAM as the beginning address of your source program and execute the BASIC commands in the BASIC mode, there is possibility that your source program might be destroyed. Therefore, it is recommended to designate the starting address of your source program at the beginning address of user's RAM plus &H100 and thereafter.

N.B.2 When you load your source program stored on the diskette, you have to designate the same address which is stored on the diskette. If the two addresses are different, the program will not be loaded correctly. This is the same in case of the cassette tape.

3-2-4 Execution of Assembler

(1) As soon as you have finished the editing of your source program. input the A command.

Example: /A [RETURN]

(The listing will appear on the screen)

or

/A, [RETURN]

(2) When you input the A, [RETURN] , only one frame of listing will be displayed. If you hit the SPACE key, the next frame of the listing appears on the screen.

When the number of errors and the prompt mark appear on the screen, this means that the assembling has been completed. Your object program is already stored in the RAM according to the ORG instruction designated by a user. Therefore, you can execute the program right away. If some errors are found, you can modify the source program by the editor commands.

(3) If you want to assemble the source program already stored on diskette, execute the A command as mentioned before.

(4) If you want to make the assemble listing, connect the printer to the machine, and input the AL command.

Example: /AL [RETURN]

If you want to stop printing halfway, type in [CTRL] + [D] keys. When you hit the [CTRL] + [D] keys, the system returns to the BASIC mode. When you execute EXEC&H5006, the system returns to the Assembler mode. The system is waiting for an editor commands.

N.B. The listing will be output on the printer under the device name of LPT0:

(5) The errors which have occurred during the assembling procedures are shown in number represented in Table 2-6-1. The assembling is continued under the conditions shown there.

(6) In order to execute the object program, return to the BASIC mode by using the E command in the Editor and execute the EXEC instruction.

Example: /E
EXEC&HXXXX [RETURN]

Or you can execute the object program by the G command in the Monitor.

Example: (In the Monitor mode)

*GXXXX [RETURN]

N.B. The program for a colour display monitor is stored at the memory locations of &H400 to &H43FF. If you use this memory area for the storage of your object program stack memory storage, the colour of the monitor will be affected. When you store your object program in this area, you have to bare this fact in mind.

3-2-5 Operation of Debugger (DEBUG70)

(1) Start the Debugger from the address of &H7000 by using the EXEC command in the BASIC.

Example: R e a d y
EXEC&H7000 [RETURN]
(The screen will be cleared)
** DEBUGGER V1.0 **
COMMAND: _
↑ Blinking Cursor

(2) This will be initiated by the G command in the Monitor

Example: (In the Monitor Mode)
*G7000 [RETURN]

(3) LOADM,,R command initiates this Debugger in the BASIC mode, just as in the Assembler.

Example: LOADM"DEBUG70",,R [RETURN]

In this case, the Debugger will be initiated right after DEBUG70 has been loaded.

S U P P L E M E N T A R Y

Even if you use the Assembler (MA-5200) together with 40KB DISK BASIC (MA-5301), it is possible to use DEBUG5A. If you load DEBUG5A in the RAM in this case, ASM9 will be destroyed. And if you load ASM9, DEBUG5A will be destroyed. Therefore, ASM9 and DEBUG5A have to be loaded according to your needs. Instead, the memory area which is occupied by DEBUG5A (&H7000 to &H7FFF), can be used for the storage of your source program or object program.

4 Simultaneous Development of Assembly and BASIC languages

4-1 Usage of Memory Area

(1) RAM Area Available to User

When you load the Assembler and Debugger in the RAM, the area available to a user is shown in Figure 1-2-1 at Section 2-1 in Chapter 1. A user has to make BASIC and Assembly programs and further their object programs in this memory area.

(2) Setting of Screen Format

When the system is equipped with the standard 32KB RAM and is used as the 80 character high-resolution screen format, the available memory area to a user for Level-3 BASIC and Level-3 DISK BASIC (memory location from the beginning of user's RAM up to the previous one byte of the Assembler working area) is 1944 bytes and 910 bytes, respectively. Therefore, it is very difficult to store BASIC and Assembly programs and their object programs in this area. In this case it is recommended to use the 40 character screen format, in order to use as many available locations as possible.

(3) Setting of Upper Address for BASIC

When the system is equipped with the standard 32KB RAM, you have to reserve the memory area for object program by using the CLEAR command in the BASIC.

(4) Effective Usage of RAM Area by Use of DEBUG5A

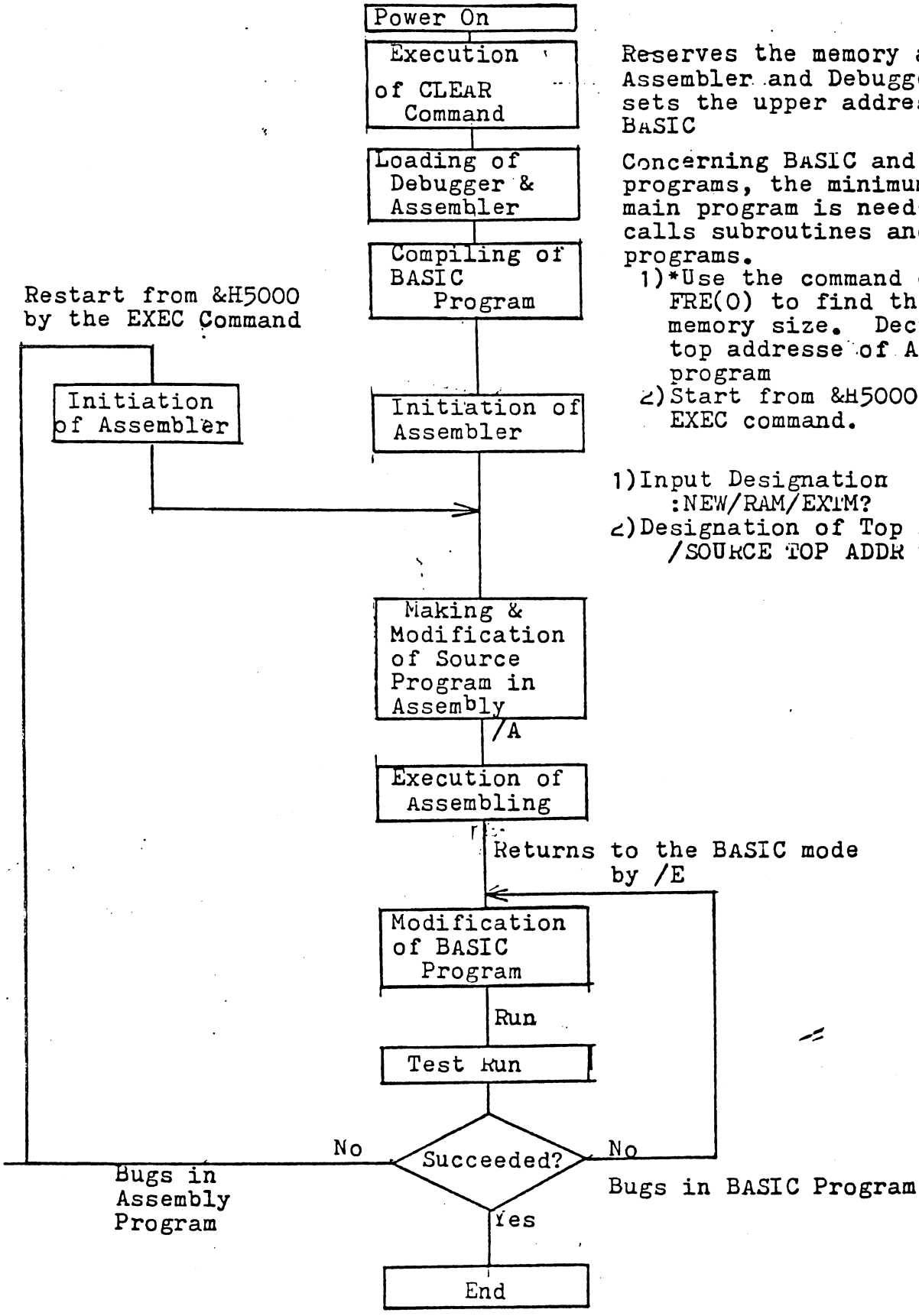
When you use a cassette tape recorder as an I/O device or 40KB DISK BASIC, it is quite possible to use DEBUG5A. When you use DEBUG5A and Assembler alternately, you can use more memory locations than DEBUG70 is used.

But, when you use a cassette tape recorder, it takes a long time to load DEBUG5A and Assembler.

(5) Others

In reference to Memory Map at Section 2-1 in Chapter 1, you can avail yourself of the machine in clever ways.

4-2. Development Procedure



Reserves the memory areas for Assembler and Debugger and sets the upper address for the BASIC

Concerning BASIC and Assembly programs, the minimum BASIC main program is needed which calls subroutines and Assembly programs.

1) *Use the command of PRINT FRE(0) to find the unused memory size. Decide the top addresse of Assembly program
2) Start from &H5000 by the EXEC command.

1) Input Designation :NEW/RAM/EXTM?
2) Designation of Top Address /SOURCE TOP ADDR ?

*The area for BASIC constants and that for the stack are not reserved if you do not run the program. Bear this in mind.

5. Assembly Language

1. Introduction

In this chapter we are going to describe the Assembly Language which is used for Microprocessor HD6809 of Basic Master Level-3, MB-6890. At Section 2 the addressing formats are describes and at Section 3 the instructions are described.

2. Address Format

There are 8 address formats as shown in the following:

1) Implied Address Format
2) Accumulator Address Format
3) Immediate Address Format
4) Direct Address Format
5) Extended Address Format (Including extended indirect address format)
6) Register Address Format
7) Indexed Address Format
   (1) Constant offset index address format
   (2) Constant offset index indirect address format
   (3) Accumulator index address format
   (4) Accumulator index indirect address format
   (5) Auto-increment address format
   (6) Auto-increment indirect address format
   (7) Auto-decrement address format
   (8) Auto-decrement indirect address format
8) Relative Address Format
   (1) Short/Long relative address format
   (2) Program counter relative address format

2-1 Implied Address Format

This is an address format in which the field to be operated is clear from an instruction. Therefore, an operand is not needed as follows:

Example: MUL. SEX

2-2 Accumulator Address Format

This is an address format in which an accumulator is operated by an instruction. You can designate Accumulator A or B as an operand of one byte. This Assembler does not allow to intert a space between the mnemonic code and the register name.

Example: DECA or DECB
TSTA or TSTB

2-3 Immediate Address Format

In this case, an operand itself becomes data. This format needs "#" at the beginning of an operand. After the "#", you have to write decimal, hexadecimal and binary numbers, and a label or an expression which is replaced by a number during the assembling procedures. As an immediate value, an instruction concerning 8-bit operation takes the value of -128 to 255, and an instruction concerning a 16-bit operation takes the value of -32768 to 65535.

Example:

STATEMENT	MACHINE CODE		
	FIRST BYTE	SECOND BYTE	THIRD BYTE
LDA#25	86	19	
LDX#&H4B01	8E	4B	01
LDD3100+600	CC	02	BC
LDA#-1	86	FF	

In Figure 5-2-1 is shown the data flow in immediate address format

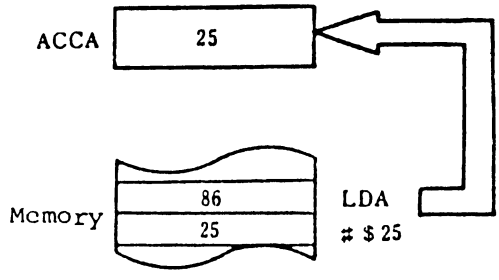


Figure 5-2-1 Data Flow In Immediate Address Format

2-4 Direct Address Format

This is address format which decide an effective address as the higher address byte from the direct page register and the lower address byte from an operand. When an operand denotes an address and the higher byte agrees with the direct page pseudo register, the Assembler selects direct address format and the lower byte as an operand. Otherwise, the Assembler selects automatically extended address format.

Or not concerning with the content of direct page pseudo register, the Assembler designates direct address format (forced direct address format). In this case, you have to add "<" at the beginning of an operand. When the higher byte disagrees with the content of direct page pseudo register and the forced direct address format is designated, this Assembler gives an error warning to give a user some attention when your program is assembled. The direct page pseudo register is to remember the address of a page consisting of 256 bytes as one page, thus covering the address space from &H0000 to &HFFFF.

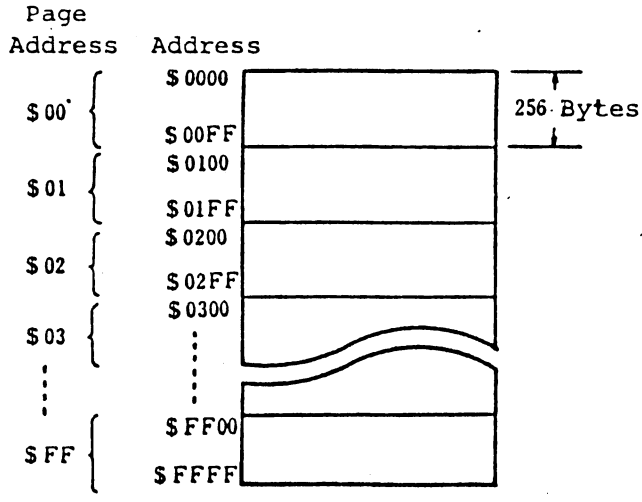


Figure 5-2-2 Relationship Between Page and Address Space

Figure 5-2-2 shows the range of direct address which can be designated by each page address. For example, if the page address is 1 which means that the content of direct address register is &H01, the direct address format can access to the memory locations of &H0100 to &H01FF.

Example:

```
                          SETDP   &H50
B6 50 00                  LDA     DATA
F6 01 00                  LDB     WK
***ERROR15
9E 00                     LDX     <WK ··· Forced Direct
(Machine Code)            ORG     &H5000
                  DATA    RMB     1
                  WK      EQU     &H0100
```

In Figure 5-2-3 is shown the data flow in direct address format

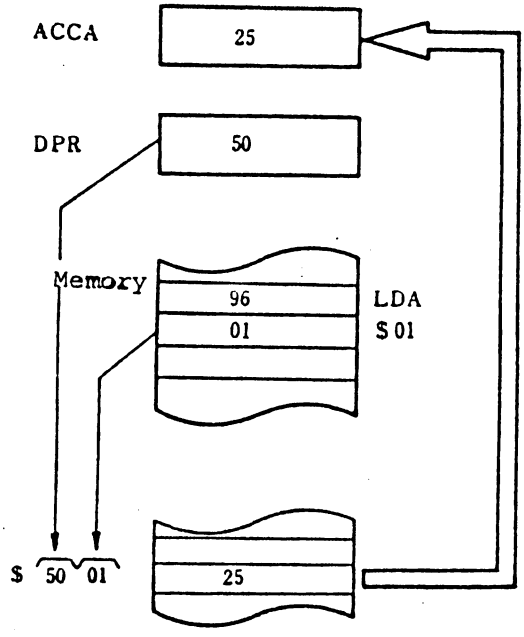


Figure 5-2-3 Data Flow in Direct Address Format

2-5 Extended Address Format

2-5-1 Extended Address Format

In case of extended address format, an operand value becomes an effective address. Even in case where direct address format is selected, it can be direct address format by putting ">" at the beginning of an operand.

Example:

STATEMENT	MACHINE CODE		
	FIRST BYTE	SECOND BYTE	THIRD BYTE
LDA &H50A0	B6	50	A0
LDX LABEL	BE	70	00
LDB>WK	F6	55	01

N.B. In the above example, it is assumed that the address of LABEL is &H7000, that the address of WK is &H5501, and that DPR (Direct Page Register) contains the value of $55.

In Figure 5-2-4 is shown the date flow in extended address format

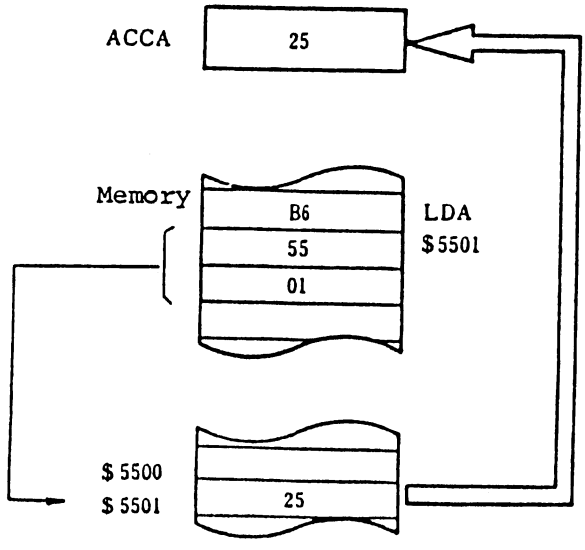


Figure 5-2-4 Data Flow in Extended Address Format

2-5-2 Extended Indirect Address Format

In case of extended indirect address format, the content of the address designated by an operand becomes an effective address. The operand has to be bracketed by "[ ]" to make this format.

Example:

STATEMENT	MACHINE CODE			
	FIRST BYTE	SECOND BYTE	THIRD BYTE	FOURTH BYTE
LDA [LABEL] DEC [WK]	A6 6A	9F 9F	50 01	10 00

N.B. In the above example, it is assumed that the address of LABEL is &H5010, that the address of WK is &H0100, and that the post byte in extended indirect address format is &H9F. As for the post byte, it will be described in detail at Section 2-7 (Index Address Format).

In Figure 5-2-5 is shown the date flow in extended indirect address format.

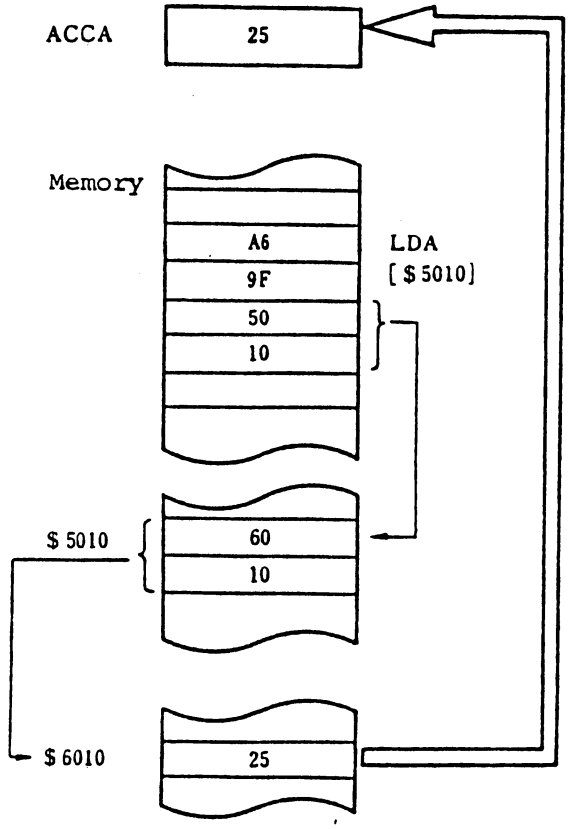


Figure 5-2-5 Data Flow in Extended Indirect Address Format

2-6 Register Address Format

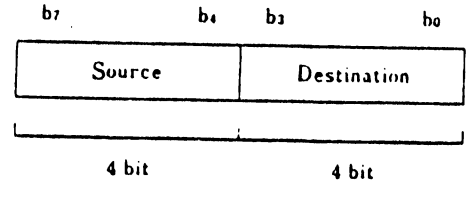
This is address format concerning the operation between registers, and a register name is designated in an operand. The instructions which can use this address format are TFR, EXG, PSHU, PULU, AND PULS. The register names designates in the operands are shown in Table 5-2-1.

Table 5-2-1 Table of Register Names

REGISTER NAME	SYMBOL	REGISTER NAME	SYMBOL
Accumulator A	A	X-Index Register	X
Accumulator B	B	Y-Index Register	Y
Double Accumulator	D	Hardware Stack Pointer	S
Condition Code Register	CC	User Stack Pointer	U
Direct Page Register	DP	Program Counter	PC

```
Example:  TFR    A, B
          EXG    X, PC
          PSHS   A, B, DP, Y
```

This operand designated by the TFR and EXG instructions is shown in Figure 5-2-6. As for these two instruction, the two registers (Source and Destination Registers) have to be either 8-bit or 16-bit ones.



0 0 0 0 ····· D	0 1 0 1 ····· P C
0 0 0 1 ····· X	1 0 0 0 ····· A
0 0 1 0 ····· Y	1 0 0 1 ····· B
0 0 1 1 ····· U	1 0 1 0 ····· C C
0 1 0 0 ····· S	1 0 1 1 ····· D P

Figure 5-2-6 Operands of TFR and EXG Instructions

The operands of PSH and PUL instructions are shown in Figure 5-2-7. If each bit is one, the corresponding register is Pushed or Pulled.

The S register (Hardware Stack Pointer) cannot be designated as an operand for PSHS and PULS instructions. The S register (User Stack Pointer) cannot be designated either as an operand for PSHU and PULU instructions.

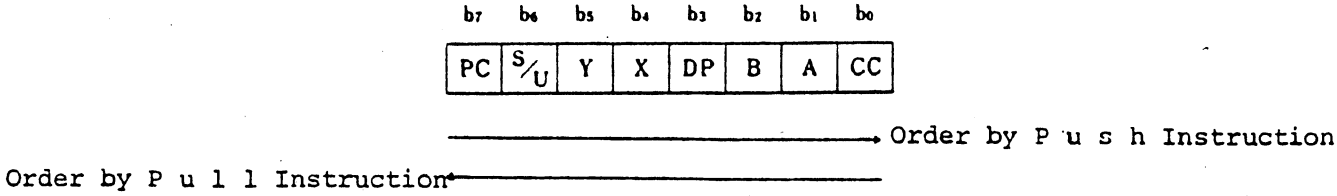


Figure 5-2-7 Operands of PSH and PUL Instructions

2-7 Index Address Format

In the HD6809 Assembly language, the 16-bit X, Y, U and S registers are called pointer registers which can be index-modified. Index address format is the one address-modified by these registers. Other than the above-mentioned registers, the program counter (PC) can be used as an index register.

There are 8 index address formats. The operand has the post byte which decides on their classifications and offset values. As for index address formats, the number of bytes in machine codes is different dependent upon the value of the offset and its size. The relationship is shown in Figure 5-2-8.

Op. Code	Post Byte	The post byte also includes 0 offset 05 bit offset
Op. Code	Post Byte	8 Bit Offset
Op. Code	Post Byte	16 Bit Offset

N.B. The op. code is one or two bytes

Figure 5-2-8 Offset and Number of Bytes

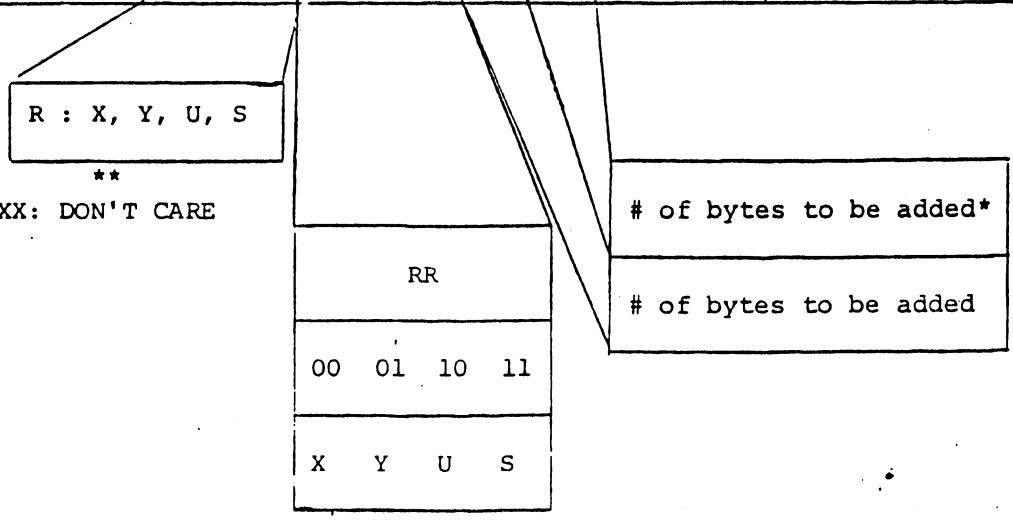
The offset value is singed one and the range of the value is shown in the following:

5 bit offset ---- -16 to 15
8 bit offset ---- -128 to 127
16 bit offset ---- -32768 to 32767

In Table 5-2-2 is shown the relationship between address format and post byte.

The Assembler will automatically decide post bytes based on the operand description, its value, and the result of the calculation.

DESCRIPTION	FORMAT	NON-INDIRECT				INDIRECT			
		DESCRIPTION	POST BYTE	+ -	+ #	DESCRIPTION	POST BYTE	+ -	+ - #
Constant Offset	0 offset	O, R	1RR00100	0	0	[0, R]	1RR10100	3	0
	5 bit offset	n, R	0RRnnnnn	1	0	—	———	–	–
	8 bit o-fset	n, R	1RR01000	1	1	[n, R]	1RR11000	4	1
	16 bit offset	n, R	1RR01001	4	2	[n, R]	1RR11001	7	2
Accumulator Offset	ACCA offset	A, R	1RR00110	1	0	[A, R]	1RR10110	4	0
	ACCB offset	B, R	1RR00101	1	0	[B, R]	1RR10101	4	0
	ACCD offset	D, R	1RR01011	4	0	[D, R]	1RR11011	7	0
Auto-incr./ Auto-decr.	1 time incr.	O, R+	1RR00000	2	0				
	2 times incr.	O, R++	1RR00001	3	0	[0, R++]	1RR10001	6	0
	1 time decr.	0, -R	1RR00010	2	0	—	———	–	–
	2 times decr.	0, --R	1RR00011	3	0	[0, --R]	1RR10011	6	0
Program Counter Relative/ Constant Offset	8 bit offset	n, PCR	1XX01100	1	1	[n, PCR]	1RR10001	6	0
		n, PC				[n, PC]			
	16 bit offset	n, PCR	1XX01101	5	2	[n, PCR]	1XX11101	8	2
		n, PC				[n, PC]			
Extended Indirect	———	—	———	–	–	[n]	10011111	5	2



* These bytes are counted as fundamental machine cycles
* The Assembler assembles these XX's as 00

2-7-1 Constant Offset Index Address Format

This format calculates an effective address by adding the operand offset value and the content of the selected pointer register. The operand is described as follows:

<Expression>, R

The R takes either of X, Y, S, U, PC, and PCR. The PCR does not mean a register, but it means that the relative value of the program counter and the expression of an operand becomes an offset value. Where an operand is Expression, R, this is describes at Section 2-8-2 (Program Counter Relative). When an offset value is 0, you can omit "0" or ",", and just designate the name of a register.

Example:

```
LAD     X)
LAD     ,X)   Either one is the same as LAD     0,X
```

N.B. This Assembler assembles the offset value as 0.

The value that an offset can take is 16 bit 2's complement number, that is, -32768 to 32767. Depending on the value of an offset, the number of bytes varies. You can add ""or"" to an operand. If you put "<" before an expression, an offset value is an 8 bit one.

Example:

```
LAD     >1,X
```

N.B. The offset value is $0001 which is of two bytes.

In Figure 5-2-9 is shown the data flow in constant offset index address format:

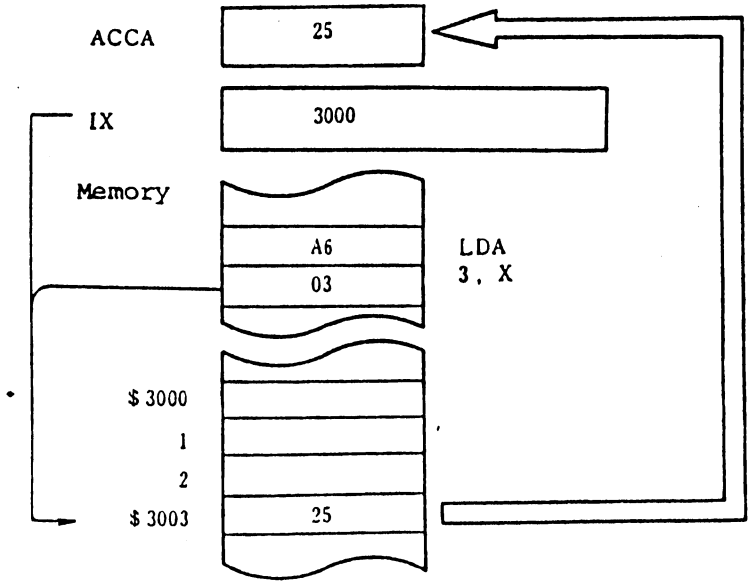


Figure 5-2-9 Data Flow In Constant Offset Index Address Format

2-7-2 Constant Offset Index Indirect Address Format

The effective address is calculated by adding the value of the selected pointer register and the offset value given by the expression of an operand. In this case, the effective address is of two bytes. To make this format, a register name is bracketed by "[ ]" in the following:

[<Expression>, R]

You can also add "<" or ">" to an operand. In Figure 5-2-10 is shown the date flow in constant offset index indirect address format.

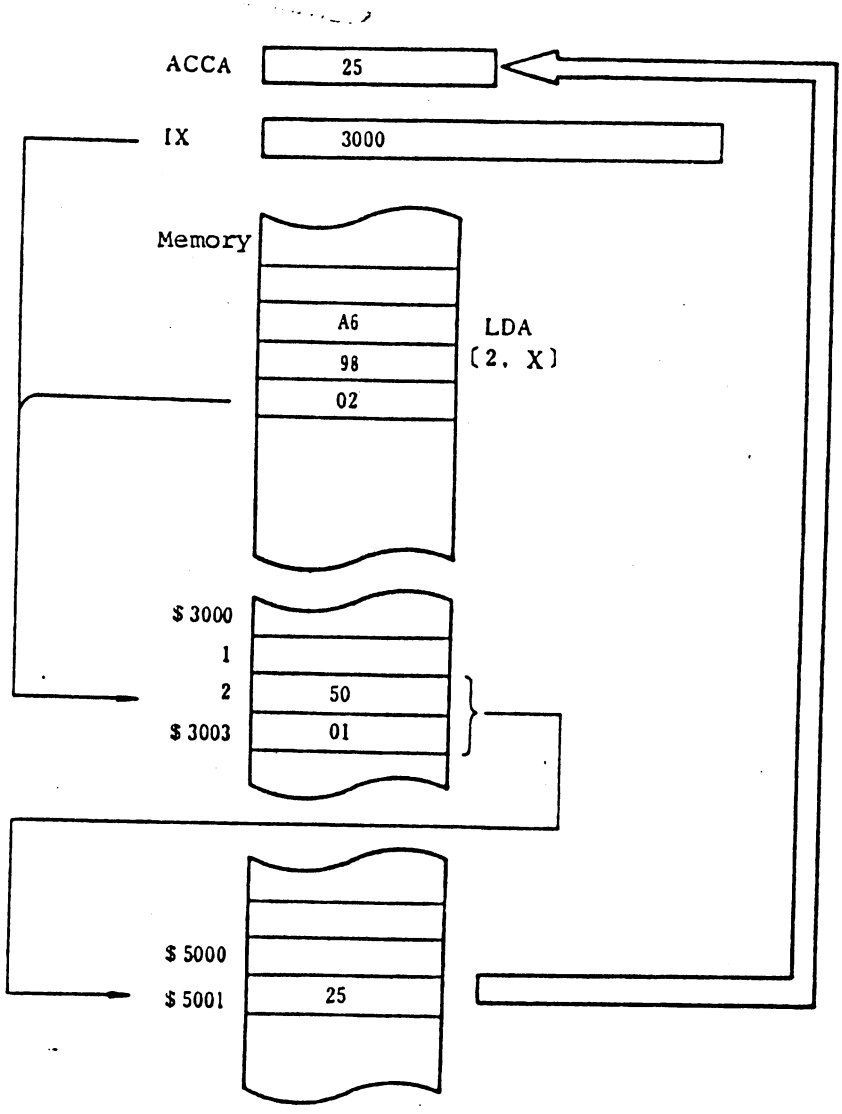


Figure 5-2-10 Data Flow in Constant Offset Index Indirect Address Format

2-7-3 Accumulator Index Address Format

The content of an accumulator is taken as an offset value. An effective address is calculated by adding the content of an accumulator and the content of a pointer register. An accumulator and a pointer register are designated in an operand.

Example:

```
LDA     A,X
CMPA    B,Y
LDD     D,U
```

A, B, or D is designated as an accumulator, X, Y, U, or S can be designated as a pointer register. In Figure 5-2-11 is shown the data flow in accumulator index address format.

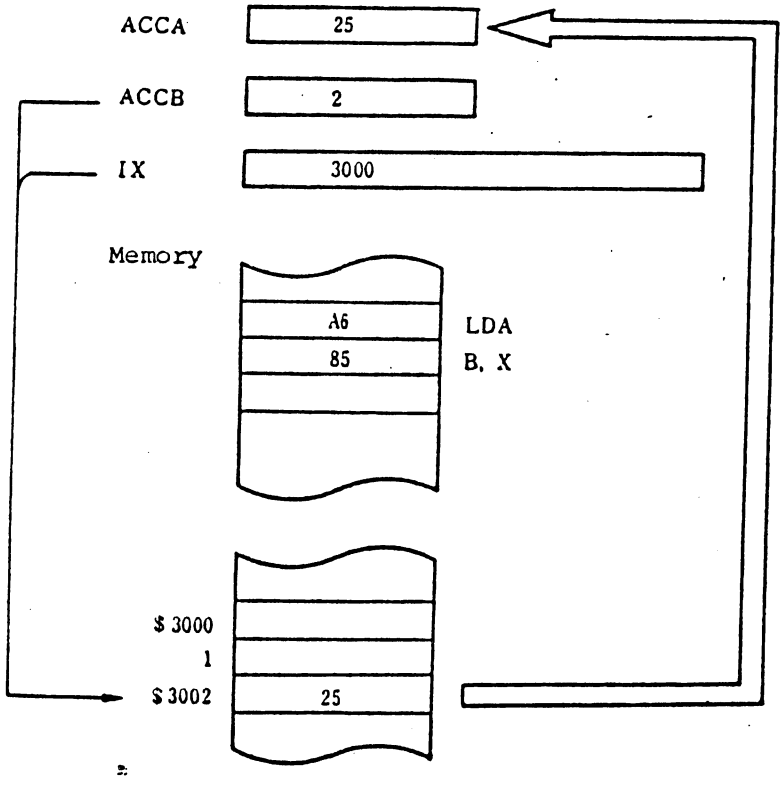


Figure 5-2-11 Data Flow in Accumulator Index Address Format

2-7-4 Accumulator Index Indirect Address Format

An effective address is calculated by adding the content of an accumulator and the content of a pointer register. The effective address is of 2 bytes. To make this address format, an operand is bracketed by "[ ]".

Example:

```
LDA     [A, X]
CMPA    [B, Y]
LDD     [D, U]
```

A, B, or D can be designated as an accumulator. X, Y, U, or S can be designated as a pointer register.

In Figure 5-2-12 is shown the data flow in accumulator index indirect address format.

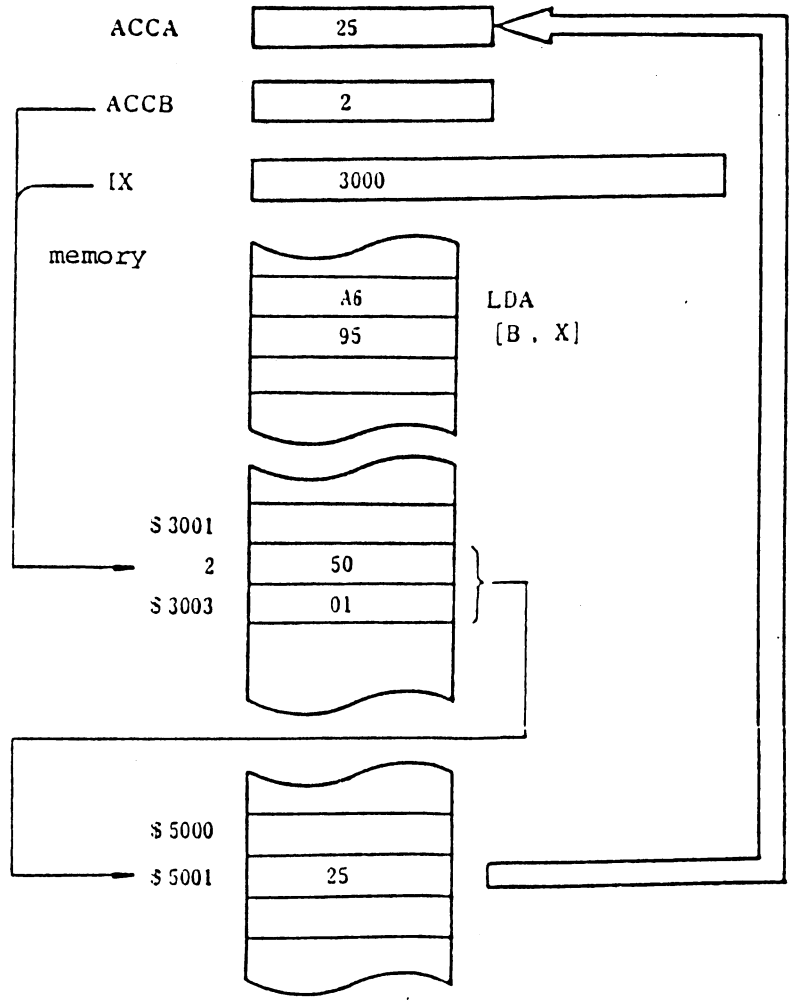


Figure 5-2-12 Data Flow in Accumulator Index Indirect Address Format

2-7-5 Auto-increment Address Format

An effective address is the content of a pointer register designated by an operand. After the execution, the content of the pointer register is incremented by 1 or 2. To make this address format, "+" or "++" is added after a register name.

Example:

```
LDA     0, X+
CMPA     , X+ .... 0 is omitted
CMPX    Y++ ..... 0 is omitted
```

N.B. In case of "+", a pointer register is incremented by 1
In case of "++", that is incremented by 2

2-7-6 Auto-increment Indirect Address Format

An effective address is the content at the location designated by a pointer register in an operand. After the execution, the pointer register is incremented by 2. To make this address format, "++" after the register name and the register name are bracketed by "[ ]". The offset value is defined as 0.

Example:

```
LDD     [0, X++]
CMPD    [ , U++] .... 0 is omitted
```

N.B. You have to add 2 +'s after a register name. Otherwise, an error takes place.

2-7-2 Auto-decrement Format

The content of a pointer register in an operand is decremented by 1 or 2. The content beocmes an effective address. To make this address format, you have to add "-" or "--" before a register name. The offset value is defined as 0.

Example:

```
LDA     0, -Y
CMPA     , -U .... 0 is omitted
ADDD      --X .... 0 is omitted
```

N.B. In case of "-", a pointer register is decremented by 1
In case of "--", that is decremented by 2

2-7-8 Auto-decrement Indirect Address Format

An effective address is the content at the location designated by the content of a pointer register minus 2. To make this address format, "--" before a register name and the register itself are bracketed by "[ ]". The offset value is defined as 0.

Example:

```
LDD     [O, __X]
CMPD
        [--U] .... O is omitted
```

N.B. You have to add 2 _'s. Otherwise, an error occurs.

2-8 Relative Address Format

2-8-1 Short/Long Branch

This is used for branching. For a short branch, an effective address is calculated by adding one byte offset value and the content of the program counter. For a long branch, that is calculated by adding two bytes offset value and the content of the program counter. The program counter contains the memory location of an instruction next to that under current execution. For short branching, jumping takes place in the relative range of -127 to 129. For long branching, jumping occurs over 64K bytes.

Example:

			Address	Machine Code	
	BEQ	LABEL	1000	27	04
	ABX		1002	3A	
	STX	DATE	1003	BF	5000
LABEL	LDA	#$FF	1006	86	FF

2-8-2 Program Counter Relative Address Format

This is relative address format concerning the program counter. The operand value is assigned as the relative value designated by an operand in reference to the content of the program counter. An effective address is calculated by adding the value assigned in an operand the content of the program counter. When you use this address format, programs become relocatable. Even if program is located anywhere in the memory, it will be executed easily. The operand is described as follows. Or you can add "<" or ">" in an operand.

<Expression>, PCR

Example:

```
JMP     &H2000, PCR
LEAU    DATA+10, PCR
LEAX    DATA   , PCR
```

In Figure 5-2-13 is shown the data flow in program counter relative address format.

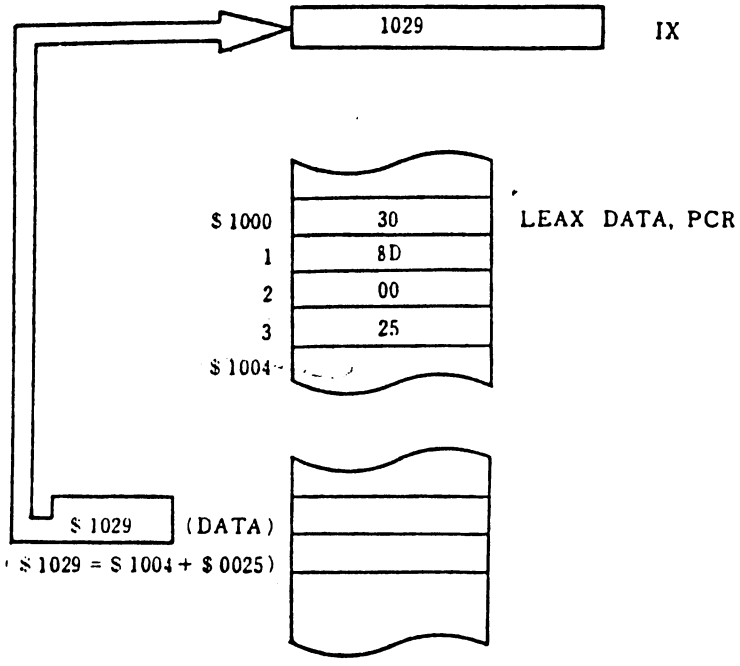


Figure 5-2-13 Data Flow in Program Counter Relative Address Format

"LEAX DATA, PCR" is the same as "LDX"DATA" shown in Figure 5-2-13. But only the different point in LEAX DATA, PCR is that the operand is relative and that program can be located anyplace in the memory very easily.

Supplementary: Expression , PC and Expression PCR are completely different when assembled. This is shown in the following:

ADDRESS	MACHINE CODE	ASSEMBLY LANGUAGE	EFFECTIVE ADDRESS
518F 5822	A68CE1	LDA $5803, PCR	5803
581F 5823	A68D5803	LDA $5803, PC	B026

3. INSTRUCTIONS AND THEIR DETAILS

Table 5-3-1 Table of HD6809 Instructions

		IMP			DIRECT			EXTND			IMMED			INDEXED			RELATIVE				7	6	5	4	3	2	1	0
		OP	~	#	OP	~	#	OP	~	#	OP	~	#	OP	~	#	OP	~	#		E	F	H	I	N	Z	V	C
ABX		3A	3	1																B ← X→X UNSIGNED	●	●	●	●	●	●	●	●
ADC	ADCA				99	4	2	B9	5	3	89	2	2	A9	4+	2+				A+M+C→A	●	●	↕	●	↕	↕	↕	↕
	ADCB				D9	4	2	F9	5	3	C9	2	2	E9	4+	2+				B+M+C→B	●	●	↕	●	↕	↕	↕	↕
ADD	ADDA				9B	4	2	BB	5	3	8B	2	2	AB	4+	2+				A+M→A	●	●	↕	●	↕	↕	↕	↕
	ADDB				DB	4	2	FB	5	3	CB	2	2	EB	4+	2+				B+M→B	●	●	↕	●	↕	↕	↕	↕
	ADDD				D3	6	2	F3	7	3	C3	4	3	E3	6+	2+				D+M:M+1→D	●	●	●	●	↕	↕	↕	↕
AND	ANDA				94	4	2	B4	5	3	84	2	2	A4	4+	2+				A∧M→A	●	●	●	●	↕	↕	R	●
	ANDB				D4	4	2	F4	5	3	C4	2	2	E4	4+	2+				B∧M→B	●	●	●	●	↕	↕	R	●
	ANDCC										1C	3	2							CC∧IMM→CC	↕	↕	↕	↕	↕	↕	↕	↕
ASL	ASLA	48	2	1																A B M C br bo 0	●	●	↕	●	↕	↕	↕	↕
	ASLB	58	2	1																	●	●	↕	●	↕	↕	↕	↕
	ASL				08	6	2	78	7	3				68	6+	2+					●	●	↕	●	↕	↕	↕	↕
ASR	ASRA	47	2	1																A B M br br C	●	●	↕	●	↕	↕	●	↕
	ASRB	57	2	1																	●	●	↕	●	↕	↕	●	↕
	ASR				07	6	2	77	7	3				67	6+	2+					●	●	↕	●	↕	↕	●	↕
BCC	BCC																24	3	2	Branch C = 0	●	●	●	●	●	●	●	●
	LBCC																10 24	5-6	4	Long Branch C = 0	●	●	●	●	●	●	●	●
BCS	BCS																25	3	2	Branch C = 1	●	●	●	●	●	●	●	●
	LBCS																10 25	5-6	4	Long Branch C = 1	●	●	●	●	●	●	●	●
BEQ	BEQ																27	3	2	Branch Z = 1	●	●	●	●	●	●	●	●
	LBEQ																10 27	5-6	4	Long Branch Z = 1	●	●	●	●	●	●	●	●
BGE	BGE																2C	3	2	Branch N ⊕ V = 0	●	●	●	●	●	●	●	●
	LBGE																10 2C	5-6	4	Long Branch N ⊕ V = 0	●	●	●	●	●	●	●	●
BGT	BGT																2E	3	2	Branch Z ∨ (N ⊕ V) = 0	●	●	●	●	●	●	●	●
	LBGT																10 2E	5-6	4	Long Branch Z ∨ (N ⊕ V) = 0	●	●	●	●	●	●	●	●
BHI	BHI																22	3	2	Branch C ∧ Z = 0	●	●	●	●	●	●	●	●
	LBHI																10 22	5-6	4	Long Branch C ∧ Z = 0	●	●	●	●	●	●	●	●
BHS	BHS																24	3	2	Branch C = 0	●	●	●	●	●	●	●	●
	LBHS																10 24	5-6	4	Long Branch C = 0	●	●	●	●	●	●	●	●
BIT	BITA				95	4	2	B5	5	3	85	2	2	A5	4+	2+				Bit Test A (M ∧ A)	●	●	●	●	↕	↕	R	●
	BITB				D5	4	2	F5	5	3	C5	2	2	E5	4+	2+				Bit Test B (M ∧ B)	●	●	●	●	↕	↕	R	●
BLE	BLE																2F	3	2	Branch Z ∨ (N ⊕ V) = 1	●	●	●	●	●	●	●	●
	LBLE																10 2F	5-6	4	Long Branch Z ∨ (N ⊕ V) = 1	●	●	●	●	●	●	●	●
BLO	BLO																25	3	2	Branch C = 1	●	●	●	●	●	●	●	●
	LBLO																10 25	5-6	4	Long Branch C = 1	●	●	●	●	●	●	●	●
BLS	BLS																23	3	2	Branch C ∨ Z = 1	●	●	●	●	●	●	●	●

		IMP			DIRECT			EXTND			IMMED			INDEXED			RELATIVE				7	6	5	4	3	2	1	0
		OP	~	#	OP	~	#	OP	~	#	OP	~	#	OP	~	#	OP	~	#		E	F	H	I	N	Z	V	C
	LBLS																10	5+6	4	Long Branch	●	●	●	●	●	●	●	●
																	23			C∨Z = 1								
BLT	BLT																2D	3	2	Branch N⊕V = 1	●	●	●	●	●	●	●	●
	LBLT																10	5+6	4	Long Branch	●	●	●	●	●	●	●	●
																	2D			N⊕V = 1								
BMI	BMI																2B	3	2	Branch N = 1	●	●	●	●	●	●	●	●
	LBMI																10	5+6	4	Long Branch	●	●	●	●	●	●	●	●
																	2B			N = 1								
BNE	BNE																26	3	2	Branch Z = 0	●	●	●	●	●	●	●	●
	LBNE																10	5+6	4	Long Branch	●	●	●	●	●	●	●	●
																	26			Z = 0								
BPL	BPL																2A	3	2	Branch N = 0	●	●	●	●	●	●	●	●
	LBPL																10	5+6	4	Long Branch	●	●	●	●	●	●	●	●
																	2A			N = 0								
BRA	BRA																20	3	2	Branch Always	●	●	●	●	●	●	●	●
	LBRA																16	5	3	Long Branch Always	●	●	●	●	●	●	●	●
BRN	BRN																21	3	2	Branch Never	●	●	●	●	●	●	●	●
	LBRN																10	5	4	Long Branch Never	●	●	●	●	●	●	●	●
																	21											
BSR	BSR																8D	7	2	Branch to Subroutine	●	●	●	●	●	●	●	●
	LBSR																17	9	3	Long Branch to Subroutine	●	●	●	●	●	●	●	●
BVC	BVC																28	3	2	Branch V = 0	●	●	●	●	●	●	●	●
	LBVC																10	5+6	4	Long Branch	●	●	●	●	●	●	●	●
																	28			V = 0								
BVS	BVS																29	3	2	Branch V = 1	●	●	●	●	●	●	●	●
	LBVS																10	5+6	4	Long Branch	●	●	●	●	●	●	●	●
																	29			V = 1								
CLR	CLRA	4F	2	1																0 → A	●	●	●	●	R	S	R	R
	CLRB	5F	2	1																0 → B	●	●	●	●	R	S	R	R
	CLR				0F	6	2	7F	7	3				6F	6+	2+				0 → M	●	●	●	●	R	S	R	R
CMP	CMPA				91	4	2	B1	5	3	81	2	2	A1	4+	2+				Compare M from A	●	●	●	●	↕	↕	↕	↕
	CMPB				D1	4	2	F1	5	3	C1	2	2	E1	4+	2+				Compare M from B	●	●	●	●	↕	↕	↕	↕
	CMPD				10	7	3	10	8	4	10	5	4	10	7+	3+				Compare M:M + 1	●	●	●	●	↕	↕	↕	↕
					93			B3			83			A3						from D								
	CMPS				11	7	3	11	8	4	11	5	4	11	7+	3+				Compare M:M + 1	●	●	●	●	↕	↕	↕	↕
					9C			BC			8C			A3						from S								
	CMPU				11	7	3	11	8	4	11	5	4	11	7+	3+				Compare M:M + 1	●	●	●	●	↕	↕	↕	↕
					93			B3			83			A3						from U								
	CMPX				9C	6	2	BC	7	3	8C	4	3	AC	6+	2+				Compare M:M + 1	●	●	●	●	↕	↕	↕	↕
																				from X								
	CMPY				10	7	3	10	8	4	10	5	4	10	7+	3+				Compare M:M + 1	●	●	●	●	↕	↕	↕	↕
					9C			BC			8C			AC						from Y								
COM	COMA	43	2	1																Ā → A	●	●	●	●	↕	↕	R	S
	COMB	53	2	1																B̄ → B	●	●	●	●	↕	↕	R	S
	COM				03	6	2	73	7	3				63	6+	2+				M̄ → M	●	●	●	●	↕	↕	R	S
CWAI											3C	20	2							CC ∧ IMM → CC;	S	↓	—	—	—	↓	—	↓
																				Wait for Interrupt								
DAA		19	2	1																Decimal Adjust A	●	●	●	●	↕	↕	↓	↕
DEC	DECA	4A	2	1																A − 1 → A	●	●	●	●	↕	↕	↕	●
	DECB	5A	2	1																B − 1 → B	●	●	●	●	↕	↕	↕	●
	DEC				0A	6	2	7A	7	3				6A	6+	2+				M − 1 → M	●	●	●	●	↕	↕	↕	●
EOR	EORA				98	4	2	B8	5	3	88	2	2	A8	4+	2+				A ⊕ M → A	●	●	●	●	↕	↕	R	●
	EORB				D8	4	2	F8	5	3	C8	2	2	E8	4+	2+				B ⊕ M → B	●	●	●	●	↕	↕	R	●
EXG	R1, R2	1E	8	2																R1 ↔ R2	↓	↓	↓	↓	↓	↓	↓	↓

		INH			DIRECT			EXTND			IMMED			INDEX-0			RELATIVE				7	6	5	4	3	2	1	0
		OP	~	#	OP	~	#	OP	~	#	OP	~	#	OP	~	#	OP	~	#		E	F	H	I	N	Z	V	C
INC	INCA	4C	2	1																A + 1 → A	●	●	●	●	↕	↕	↕	●
	INCB	5C	2	1																B + 1 → B	●	●	●	●	↕	↕	↕	●
	INC				0C	6	2	7C	7	3				6C	6+	2+				M + 1 → M	●	●	●	●	↕	↕	↕	●
JMP					0E	3	2	7E	4	3				6E	3+	2+				EA → PC	●	●	●	●	●	●	●	●
JSR					9D	7	2	BD	8	3				AD	7+	2+				Jump to Subroutine	●	●	●	●	●	●	●	●
LD	LDA				96	4	2	B6	5	3	86	2	2	A6	4+	2+				M → A	●	●	●	●	↕	↕	R	●
	LDB				D6	4	2	F6	5	3	C6	2	2	E6	4+	2+				M → B	●	●	●	●	↕	↕	R	●
	LDD				DC	5	2	FC	6	3	CC	3	3	EC	5+	2+				M:M + 1 → D	●	●	●	●	↕	↕	R	●
	LDS				10	6	3	10	7	4	10	4	4	10	6+	3+				M:M + 1 → S	●	●	●	●	↕	↕	R	●
					DE			FE			CE			EE															
	LDU				DE	5	2	FE	6	3	CE	3	3	EE	5+	2+				M:M + 1 → U	●	●	●	●	↕	↕	R	●
	LDX				9E	5	2	BE	6	3	8E	3	3	AE	5+	2+				M:M + 1 → X	●	●	●	●	↕	↕	R	●
	LDY				10	6	3	10	7	4	10	4	4	10	6+	3+				M:M + 1 → Y	●	●	●	●	↕	↕	R	●
					9E			BE			8E			AE															
LEA	LEAS													32	4+	2+				EA → S	●	●	●	●	●	●	●	●
	LEAU													33	4+	2+				EA → U	●	●	●	●	●	●	●	●
	LEAX													30	4+	2+				EA → X	●	●	●	●	↕	●	●	●
	LEAY													31	4+	2+				EA → Y	●	●	●	●	↕	●	●	●
LSL	LSLA	48	2	1																A / B / M: C b7 b0	●	●	●	●	↕	↕	↕	↕
	LSLB	58	2	1																	●	●	●	●	↕	↕	↕	↕
	LSL				08	6	2	78	7	3				68	6+	2+					●	●	●	●	↕	↕	↕	↕
LSR	LSRA	44	2	1																A / B / M: 0 → b7 b0 C	●	●	●	●	R	↕	●	↕
	LSRB	54	2	1																	●	●	●	●	R	↕	●	↕
	LSR				04	6	2	74	7	3				64	6+	2+					●	●	●	●	R	↕	●	↕
MUL		3D	11	1																A × B → D Unsigned	●	●	●	●	●	↕	●	↕
NEG	NEGA	40	2	1																A + 1 → A	●	●	↕	●	↕	↕	↕	↕
	NEGB	50	2	1																B + 1 → B	●	●	↕	●	↕	↕	↕	↕
	NEG				00	6	2	70	7	3				60	6+	2+				M + 1 → M	●	●	↕	●	↕	↕	↕	↕
NOP		12	2	1																No Operation	●	●	●	●	●	●	●	●
OR	ORA				9A	4	2	BA	5	3	8A	2	2	AA	4+	2+				A OR M → A	●	●	●	●	↕	↕	R	●
	ORB				DA	4	2	FA	5	3	CA	2	2	EA	4+	2+				B OR M → B	●	●	●	●	↕	↕	R	●
	ORCC										1A	3	2							CC OR IMM → CC					↕	↕	↕	↕
PSH	PSHS	34	5+i	2																Push Registers on S Stack	●	●	●	●	●	●	●	●
	PSHU	36	5+i	2																Push Registers on U Stack	●	●	●	●	●	●	●	●
PUL	PULS	35	5+i	2																Pull Registers from S Stack				↕				
	PULU	37	5+i	2																Pull Registers from U Stack				↕				
ROL	ROLA	49	2	1																A / B / M: C b7 b0	●	●	●	●	↕	↕	↕	↕
	ROLB	59	2	1																	●	●	●	●	↕	↕	↕	↕
	ROL				09	6	2	79	7	3				69	6+	2+					●	●	●	●	↕	↕	↕	↕
ROR	RORA	46	2	1																A / B / M: C b7 b0	●	●	●	●	↕	↕	↕	↕
	RORB	56	2	1																	●	●	●	●	↕	↕	↕	↕
	ROR				06	6	2	76	7	3				66	6+	2+					●	●	●	●	↕	↕	↕	↕
RTI		3B	6/15	1																Return from Interrupt				↕				
RTS		39	5	1																Return from Subroutine	●	●	●	●	●	●	●	●
SBC	SBCA				92	4	2	B2	5	3	82	2	2	A2	4+	2+				A - M - C → A	●	●	↕	●	↕	↕	↕	↕
	SBCB				D2	4	2	F2	5	3	C2	2	2	E2	4+	2+				B - M - C → B	●	●	↕	●	↕	↕	↕	↕

		IMP			DIRECT			EXTND			IMMED			INDEX②			RELATIVE				7	6	5	4	3	2	1	0
		OP	~	=	OP	~	=	OP	~	=	OP	~	=	OP	~	=	OP	~	=		E	F	H	I	N	Z	V	C
SEX		1D	2	1																Sign Extend B into A (B∩bit7=1 FF→A (B∩bit7=0 0→A	●	●	●	●	1	↕	●	●
ST	STA				97	4	2	B7	5	3				A7	4+	2+				A→M	●	●	●	●	↕	↕	R	●
	STB				D7	4	2	F7	5	3				E7	4+	2+				B→M	●	●	●	●	↕	↕	R	●
	STD				DD	5	2	FD	6	3				ED	5+	2+				D→M:M+1	●	●	●	●	↕	↕	R	●
	STS				10 DF	6	3	10 FF	7	4				10 EF	6+	3+				S→M:M+1	●	●	●	●	↕	↕	R	●
	STU				DF	5	2	FF	6	3				EF	5+	2+				U→M:M+1	●	●	●	●	↕	↕	R	●
	STX				9F	5	2	BF	6	3				AF	5+	2+				X→M:M+1	●	●	●	●	↕	↕	R	●
	STY				10 9F	6	3	10 BF	7	4				10 AF	6+	3+				Y→M:M+1	●	●	●	●	↕	↕	R	●
SUB	SUBA				90	4	2	B0	5	3	80	2	2	A0	4+	2+				A−M→A	●	●	↕	●	↕	↕	↕	↕
	SUBB				D0	4	2	F0	5	3	C0	2	2	E0	4+	2+				B−M→B	●	●	↕	●	↕	↕	↕	↕
	SUBD				93	6	2	B3	7	3	83	4	3	A3	6+	2+				D−M:M+1→D	●	●	●	●	↕	↕	↕	↕
SWI	SWI①③	3F	19	1																Software interrupt 1	S	S	●	S	●	●	●	●
	SWI 2④	10 3F	20	2																Software interrupt 2	S	●	●	●	●	●	●	●
	SWI 3⑤	11 3F	20	2																Software interrupt 3	S	●	●	●	●	●	●	●
SYNC		13	≥2	1																Synchronize to interrupt	●	●	●	●	●	●	●	●
TFR	R1, R2	1F	7	2																R1→R2⑨				⑪				
TST	TSTA	4D	2	1																Test A	●	●	●	●	↕	↕	R	●
	TSTB	5D	2	1																Test B	●	●	●	●	↕	↕	R	●
	TST				0D	6	2	7D	7	3				6D	6+	2+				Test M	●	●	●	●	↕	↕	R	●

N.B. Extended indirect and program counter relative commands will be described in "Index" column.

SYMBOLS AND ABBREVIATIONS

(1) Operation Symbols

= Transfer Direction, Exclusive Or, = AND, += Arithmetic Addition, =OR, -= Arithmetic subtraction.

(2) Abbreviation Symbols

OP = Operation Code
= Number of Cycles of MPU
= Number of bytes of Instruction

(3) Memory and Address Format

M = Memory location
M+1 = Memory location plus 1
IMM = Immediate value
ACCM = Accumulator Address Format
REG = Register Address Format
DIRECT = Direct Address Format
EXTND = Extended Address Format
IMMED = Immediate Address Format
INDEX = Index Address Format
RELATIVE = Relative Address Format

(4) Condition Code Register

E(b7) = Entire Flag
F(b6) = FIRQ Master Flag
H(b5) = Carry Flag from bit 3 to bit 4

(4) Condition Code Register (continued)

I(b4) = IRQ Master Flag
N(b3) = Negative Flag
Z(b2) = Zero Flag
V(b1) = Overflow Flag in case of 2' Complement
C(b0) = Carry Flag From bit 7 or Borrow Flag to bit 7

(5) Symbols to Change the Content of Condition Code Register

R = Set
S = Reset
= Set if the condition is satisfied. Otherwise, rest.
= Unchanged

(6) Notes on Condition Code Register

(1) = In case of index mode, the necessary machine cycles and number of bytes in this Table to the values shown in Table 5-3-2.

(2) = In case of EXG and TFR instructions, the R1 and R2 registers are either of 8 bits or of 16 bits. 8 bit registers:A, B, CC, DP and 16 bit registers:X, Y, U, S, D, PC.

(3) = EA stands for effective address.

(4) = Instruction cycles for PSH/PUL
This is calculated by adding the machine cycles shown in the table to the number of bytes of the register to be PUSHed or PULLed.

(5) = 5(6)-----5 cycles for non-branching
6 cycles for branching

(6) SWI sets the I and F flags. SWI2 and SWI3 unchanges I and F flags.

(7) The condition code is set by the result of an instruction executed.

(8) = After this instruction has been executed, the H and V flags does not have any meaning.

(9) = If b7 of the ACCB is 1 for the MUL instruction, the C flag will be set.

(10) = If the CC is designated in an operand, this will be set according to the result of the instruction executed. But in case of the TFR instruction this will be set if the CC is designated in the destination.

e 5-3-2 Post Bytes of Indexed, Program Counter Relative, and Extended Indirect Modes

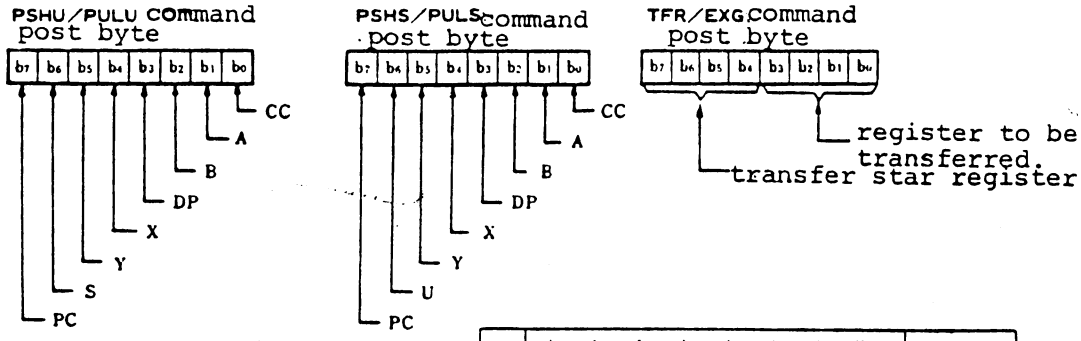
ADDRESSING MODE			NON-INDIRECT				DIRECT			
			ASSEMBLER FORMAT	POST BYTE	+ -	+ #	ASSEMBLER FORMAT	POST BYTE	# -	+ #
	Constant Offset	Zero Offset	O, R	1RR00100	0	0	[O, R]	1RR10100	3	0
		5 bit Offset	n, R	0RRnnnnn	1	0	——	————	-	—
		8 bit Offset	n, R	1RR01000	1	1	[n, R]	1RR11000	4	1
		16 bit Offset	n, R	1RR01001	4	2	[n, R]	1RR11001	7	2
	Accumulator Offset	A Reg. Offset	A, R	1RR00110	1	0	[A, R]	1RR10110	4	0
		B Reg. Offset	B, R	1RR00101	1	0	[B, R]	1RR10101	4	0
		D Reg. Offset	D, R	1RR01011	4	0	[D, R]	1RR11011	7	0
DEXED	Auto Incr. /Auto Dec.	Increment(+1)	O, R+	1RR00000	2	0	——	————	—	—
		Increment(+2)	O, R++	1RR00001	3	0	[O, R++]	1RR10001	6	0
		Decrement(-1)	O, -R	1RR00010	2	0				
		Decrement(-2)	O, --R	1RR00011	3	0	[O, --R]	1RR10011	6	0
gram nter atives/ stant set	Program Counter Relative	8 bit Offset	n, PCR n, PC	1XX01100	1	1	[n, PCR] [n, PC]	1XX11100	4	1
	Relative/ Constant Offset	16 bit Offset	n, PCR n, PC	1XX01101	5	2	[n, PCR] [n, PC]	1XX11101	8	2
ended indirect		16 bit Offset	——	————	—	—	[n]	10011111	5	2

reviations:

RR: 00=X X=Don't care
01=Y (The Assembler understands that X is O.)
10=U
11=S

$\overset{+}{-}$: Number of bytes to be added to the fundamental cycles.

$\overset{+}{\#}$: Number of bytes to be added to the basic machine cycles.



	b7 b3	b6 b2	b5 b1	b4 / b0	regist.
0	0	0	0	0	D
1	0	0	0	1	X
2	0	0	1	0	Y
3	0	0	1	1	U
4	0	1	0	0	S
5	0	1	0	1	PC
8	1	0	0	0	A
9	1	0	0	1	B
A	1	0	1	0	CC
B	1	0	1	1	DP

Table 5-3-3 Quick Conversion TAble between Mnemonic and Machine Codes

上位 / 下位	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	NEG (DIR)			COM (DIR)	LSR (DIR)		ROR (DIR)	ASR (DIR)	ASL·LSL (DIR)	ROL (DIR)	DEC (DIR)		INC (DIR)	TST (DIR)	JMP (DIR)	CLR (DIR)
1	*	**	NOP (IMP)	SYNC (IMP)			LBRA (REL)	LBSR (REL)		DAA (ACC)	OR (CC) (IMM)		AND (CC) (IMM)	SEX (IMP)	EXG (REG)	TFR (REG)
2	BRA (REL)	BRN (REL)	BHI (REL)	BLS (REL)	BCC BHS (REL)	BCS BLO (REL)	BNE (REL)	BEQ (REL)	BVC (REL)	BVS (REL)	BPL (REL)	BMI (REL)	BGE (REL)	BLT (REL)	BGT (REL)	BLE (REL)
3	LEA (X) (IND)	LEA (Y) (IND)	LEA (S) (IND)	LEA (U) (IND)	PSH (S) (REG)	PUL (S) (REG)	PSH (U) (REG)	PUL (U) (REG)		RTS (IMP)	ABX (IMP)	RTI (IMP)	CWAI (IMP)	MUL (IMP)		SWI (IMP)
4	NEG (A) (ACC)			COM (A) (ACC)	LSR (A) (ACC)		ROR (A) (ACC)	ASR (A) (ACC)	ASL·LSL(A) (ACC)	ROL (A) (ACC)	DEC (A) (ACC)		INC (A) (ACC)	TST (A) (ACC)		CLR (A) (ACC)
5	NEG (B) (ACC)			COM (B) (ACC)	LSR (B) (ACC)		ROR (B) (ACC)	ASR (B) (ACC)	ASL·LSL(B) (ACC)	ROL (B) (ACC)	DEC (B) (ACC)		INC (B) (ACC)	TST (B) (ACC)		CLR (B) (ACC)
6	NEG (IND)			COM (IND)	LSR (IND)		ROR (IND)	ASR (IND)	ASL·LSL (IND)	ROL (IND)	DEC (IND)		INC (IND)	TST (IND)	JMP (IND)	CLR (IND)
7	NEG (EXT)			COM (EXT)	LSR (EXT)		ROR (EXT)	ASR (EXT)	ASL·LSL (EXT)	ROL (EXT)	DEC (EXT)		INC (EXT)	TST (EXT)	JMP (EXT)	CLR (EXT)
8	SUB (A) (IMM)	CMP (A) (IMM)	SBC (A) (IMM)	SUB (D) (IMM)	AND (A) (IMM)	BIT (A) (IMM)	LD (A) (IMM)		EOR (A) (IMM)	ADC (A) (IMM)	OR (A) (IMM)	ADD (A) (IMM)	CMP (X) (IMM)	BSR (REL)	LD (X) (IMM)	
9	SUB (A) (DIR)	CMP (A) (DIR)	SBC (A) (DIR)	SUB (D) (DIR)	AND (A) (DIR)	BIT (A) (DIR)	LD (A) (DIR)	ST (A) (DIR)	EOR (A) (DIR)	ADC (A) DIR)	OR (A) (DIR)	ADD (A) (DIR)	CMP (X) (DIR)	JSR (DIR)	LD (X) (DIR)	ST (X) (DIR)
A	SUB (A) (IND)	CMP (A) (IND)	SBC (A) (IND)	SUB (D) (IND)	AND (A) (IND)	BIT (A) (IND)	LD (A) (IND)	ST (A) (IND)	EOR (A) (IND)	ADC (A) (IND)	OR (A) (IND)	ADD (A) (IND)	CMP (X) (IND)	JSR (IND)	LD (X) (IND)	ST (X) (IND)
B	SUB (A) (EXT)	CMP (A) (EXT)	SBC (A) (EXT)	SUB (D) (EXT)	AND (A) (EXT)	BIT (A) (EXT)	LD (A) (EXT)	ST (A) (EXT)	EOR (A) (EXT)	ADC (A) (EXT)	OR (A) (EXT)	ADD (A) (EXT)	CMP (X) (EXT)	JSR) (EXT)	LD (X) (EXT)	ST (X) (EXT)
C	SUB (B) (IMM)	CMP (B) (IMM)	SBC (B) (IMM)	ADD (D) (IMM)	AND (B) (IMM)	BIT (B) (IMM)	LD (B) (IMM)		EOR (B) (IMM)	ADC (B) (IMM)	OR (B) (IMM)	ADD (B) (IMM)	LD (D) (IMM)		LD (U) (IMM)	
D	SUB (B) (DIR)	CMP (B) (DIR)	SBC (B) (DIR)	ADD (D) (DIR)	AND (B) (DIR)	BIT (B) (DIR)	LD (B) (DIR)	ST (B) (DIR)	EOR (B) (DIR)	ADC (B) (DIR)	OR (B) (DIR)	ADD (B) (DIR)	LD (D) (DIR)	ST (D) (DIR)	LD (U) (DIR)	ST (U) (DIR)
E	SUB (B) (IND)	CMP (B) (IND)	SBC (B) (IND)	ADD (D) (IND)	AND (B) (IND)	BIT (B) (IND)	LD (B) (IND)	ST (B) (IND)	EOR (B) (IND)	ADC (B) (IND)	OR (B) (IND)	ADD (B) (IND)	LD (D) (IND)	ST (D) (IND)	LD (U) (IND)	ST (U) (IND)
F	SUB (B) (EXT)	CMP (B) (EXT)	SBC (B) (EXT)	ADD (D) (EXT)	AND (B) (EXT)	BIT (B) (EXT)	LD (B) (EXT)	ST (B) (EXT)	EOR (B) (EXT)	ADC (B) (EXT)	OR (B) (EXT)	ADD (B) (EXT)	LD (D) (EXT)	ST (D) (EXT)	LD (U) (EXT)	ST (U) (EXT)

DIR=Direct Addressing
EXT=Extended Addressing
IMM=Immediate Addressing
IND=Indexed Addressing
IMP=Implied Addressing
REL=Relative Addressing
REG=Register Addressing
ACC=Accumulator Addressing

A=Accumulator A
B=Accumulator B
D=Double Accumulator
X=X Index Register
Y=Y Index Register
S=Hardware Stack Pointer Register
U=User Stack Pointer Register
CC=Condition Code Register

* Refer to No. 2
** Refer to No. 3

Table 5-3-3 Quick Conversion Table between Mnemonic and Machine Codes

All the instructions shown below of the first byte being $10

上位 \ 下位	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0																
1																
2		LBRN (REL)	LBHI (REL)	LBLS (REL)	LBHS LBCC (REL)	LBCS LBLO (REL)	LBNE REL.	LBEQ (REL.	LBVC (REL)	LBVS REL.	LBPL (REL.	LBMI REL)	LBGE REL.	LBLT (REL)	LBGT (REL)	LBLE (REL.
3																SWI2 (IMP)
4																
5																
6																
7																
8				CMP (D) (IMM)									CMP (Y) (IMM)		LD (Y) (IMM)	
9				CMP (D) DIR)									CMP (Y) (DIR)		LD (Y) (DIR)	ST (Y) (DIR)
A				CMP (D) (IND)									CMP (Y) (IND)		LD (Y) (IND)	ST (Y) (IND)
B				CMP (D) (EXT)									CMP (Y) (EXT)		LD (Y) (EXT)	ST (Y) (EXT)
C															LD (S) (IMM)	
D															LD (S) (DIR)	ST (S) (DIR)
E															LD (S) (IND)	ST (S) (IND)
F															LD (S) (EXT)	ST (S) (EXT)

Table 5-3-3 Quick Conversion Table between Mnemonic and Machine Codes

All the instructions shown below of the first byte being $11

LSN \ MSN	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0																
1																
2																
3																SWI3 (IMP)
4																
5																
6																
7																
8				CMP (U) (IMM)									CMP (S) (IMM)			
9				CMP (U) DIR)									CMP (S) (DIR)			
A				CMP (U) IND)									CMP (S) (IND)			
B				CMP (U) (EXT)									CMP (S) (EXT)			
C																
D																
E																
F																

A P P E N D I X

MODIFICATION OF THE SOURCE TOP ADDRESS OF YOUR SOURCE PROGRAM

When you save your source program made by the Assembler on cassette or diskette by using the S command, the system saves the source top address other than the content of your source program.

Your source program will be managed under the file name and the source top address. Therefore, when you load your source program into the RAM from cassette tape or diskette, you have to designate the source top address and the file name.

If the source top address type in through the keyboard is different from that recorded on cassette tape or diskette, there is no guarantee that your source program will be loaded correctly. You had better write down the file name and the source top address. And you have to designate the same file name and source top address when you load your source program.

Because your source program is managed under the file name and the source top address, you cannot load into the different memory locations by modifying your source top address once saved on cassette tape or diskette. As shown in the following example, you can load your source program at the different memory locations from those recorded on cassette tape or diskette.

Example:

For example set the screen mode of Basic Master Level-3 to the 80 character mode (normal mode). the system has to be set in the Assembler mode. You have to designate the source top address as &H2000 through the keyboard and to make your source program. And after that you have to save this source program under the file name of TEST on cassette tape or diskette. In this case the source program will be managed in the following:

(1) Source Top Address: &H2000
(2) File Name: "TEST"

In case this source program is loaded at the memory loaction of &H3000 and thereafter, the following procedures are needed:

(In The Assembler Mode)

*ASMB/EDITR_V2.0

(1) :NEW/RAM/EXTM? [N] --- You have to type in the [N] key.

(2) /SOURCE TOP ADDR? 3000 [RETURN]
↑ Designate the Source Top Addr.

(3) / [E] --- You have to key in E while the system is waiting for an editor command. (This instruction returns the system to the BASIC mode.)

(4) LOADM" Device Name : TEST", &H1000 [RETURN]

↑ Refer to * to calculate the offset value

(This instruction loads your source program under the file name of "TEST" at the memory lcoation of &H3000 and thereafter).

(5) EXEC&H5000

(This instruction puts the system in the Assembler mode.)

*ASMB/EDITR__V2.0

(6) :NEW/RAM/EXTM? [R] --- Type in the [R] key.

/ _ ←----------The system is waiting for an editor command

↑ Blinking Cursor

(It is quite possible to compile your source program by editor commands.)

The source program, the source top address of which has been changed from &H2000 to &H3000 has to be saved on cassette tape or diskette by the S command in the Editor under the file name of TEST. The file will be managed under:

(1) Source Top Address: &H3000

(2) File Name: TEST

* Method of Calculating Offset Value:

Offset Value = Source Top Address Designated by Keying in - Source Top Address of your Source Program to be loaded

In the above example, the offset value is:

&H1000 = &H3000 - &H2000